Jadex

Tool Guide

Release 0.941
01. Dezember 2005
http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

Alexander Pokahr
Lars Braubach
Rüdiger Leppin
Andrzej Walczak

Distributed Systems Group
University of Hamburg, Germany
http://vsis-www.informatik.uni-hamburg.de

If you have support questions about Jadex please use the sourceforge help forum and mailing list for that purpose (available at http://sourceforge.net/projects/jadex/).

Table of Contents

1. Introduction 1
2. Jadex Control Center 3
  2.1. Using the JCC 3
  2.2. Platform Settings 4
3. Jadex Starter 7
  3.1. Model Tree 7
  3.2. Running Agents 8
  3.3. Model Panel 8
4. Introspector 11
  4.1. Base Panels 11
  4.2. Debugger Panel 14
5. Conversation Center 17
  5.1. Sending Messages 17
    5.1.1. Agent Selector Dialog 18
  5.2. Receiving Messages 19
6. BDI Tracer 21
  6.1. Main Window 21
    6.1.1. Trace Tree 22
    6.1.2. Trace Table 23
    6.1.3. Trace Exploration Graph 23
  6.2. Menus 23
    6.2.1. Agent Menu 23
    6.2.2. Table Menu 24
    6.2.3. Graph Menu 25
  6.3. Agent Filter Dialog 26
7. Beanynizer 27
  7.1. Installation 27
  7.2. Creating an Ontology 27
  7.3. Ontology Options 28
  7.4. Class Options 29
  7.5. Slot Options 30
  7.6. Converting an Existing Ontology 32
  7.7. Final Notes 32
8. Jadexdoc Tool 33
  8.1. Usage 33
  8.2. Source Files 35
  8.3. Generated Files 37
  8.4. Documentation Comments 37
  8.5. Options 38
    8.5.1. Jadexdoc Options 38
    8.5.2. Options Provided by the Standard Doclet 39
A. Jadex Remote Monitoring Agent 45
  A.1. Start New Agent Dialog 45
  A.2. Loading Agents from Jar Files 46
B. Logger 49
  B.1. Logging Overview 49
  B.2. Generate Log Messages 49
  B.3. View Log Messages 50

Bibliography 55

Chapter 1. Introduction

Jadex includes various tools for runtime and debugging activities as well as for development and documentation purposes. There are also some legacy tools developed with Jadex for the JADE platform.

Jadex Runtime Tools.

- Jadex Control Center. The Control Center is the central access point for all runtime tools. It offers functionalities provided by plug-ins in separate perspectives.
- Agent Starter. The Starter plug-in offers a user-interface to administer the agents on the platform. It can be used to load, start and kill selected agents.
- Jadex Introspector. The Introspector plug-in can be used to observe internal state of agents including their beliefs, goals and plans. It also includes a debugger that allows to execute agents stepwise.
- Tracer Agent. The Tracer plug-in may be used to visualize the internal processes of an agent at runtime and show causal dependencies among agent's beliefs, goal, and plans.
- Conversation Center. The Conversation Center can be used to compose messages in a user interface and send them to agents directly.

Jadex Development Tools.

- Beanynizer. The Protégé™ plugin is handy with creating whole ontologies and converting them to Java™ bean classes.
- Jadexdoc. The documentation tool helps to create JavaDoc-like documentation for Jadex agents.

JADE-Specific Runtime Tools.

- Jadex RMA The Remote Monitoring Agent can be used on the Jade platform to start and manage Jadex agents and other Jade-specific tools. From the Jadex RMA it is possible to start the Jadex Control Center per button-click if one wants to use the default Jadex runtime tools.
- Logger Agent. The Logger collects, processes and visualizes all data send by agents using the `java.util.logging` API. It has been debeloped using JADE platform facilities and is therefore currently unavailble for the Standalone (and other) platforms.

Chapter 2. Jadex Control Center

The Jadex Control Center (JCC) represents the main access point for all available Jadex runtime tools. The JCC itself provides its functionalities via plugins and is therefore quite easily extensible. Currently the following plugins are shipped with the standard distribution of Jadex: Starter, Introspector, Conversation Center, and Tracer. Each tool provides its own perspective in the JCC and is described in subsequent chapters.

Main aspect of the JCC is the project handling. A project is used to store user settings made in the JCC itself (ike window sizes or user settings) and the settings from the various plugins. Project files consist of a main project file (ending ".jpr" for Jadex project) and additional property files for each plugin. In addition, the JCC uses a startup configuration, taken from the file `jcc.properties` which is by default located in the user Jadex configuration directory (`%USERPROFILE%/jadex` on MS Windows™, `$HOME/.jadex` on other operating systems). This properties contain a list of used plugins as well as a pointer to the last project the user worked with. At the launch time of JCC the last project will be automatically reopened.

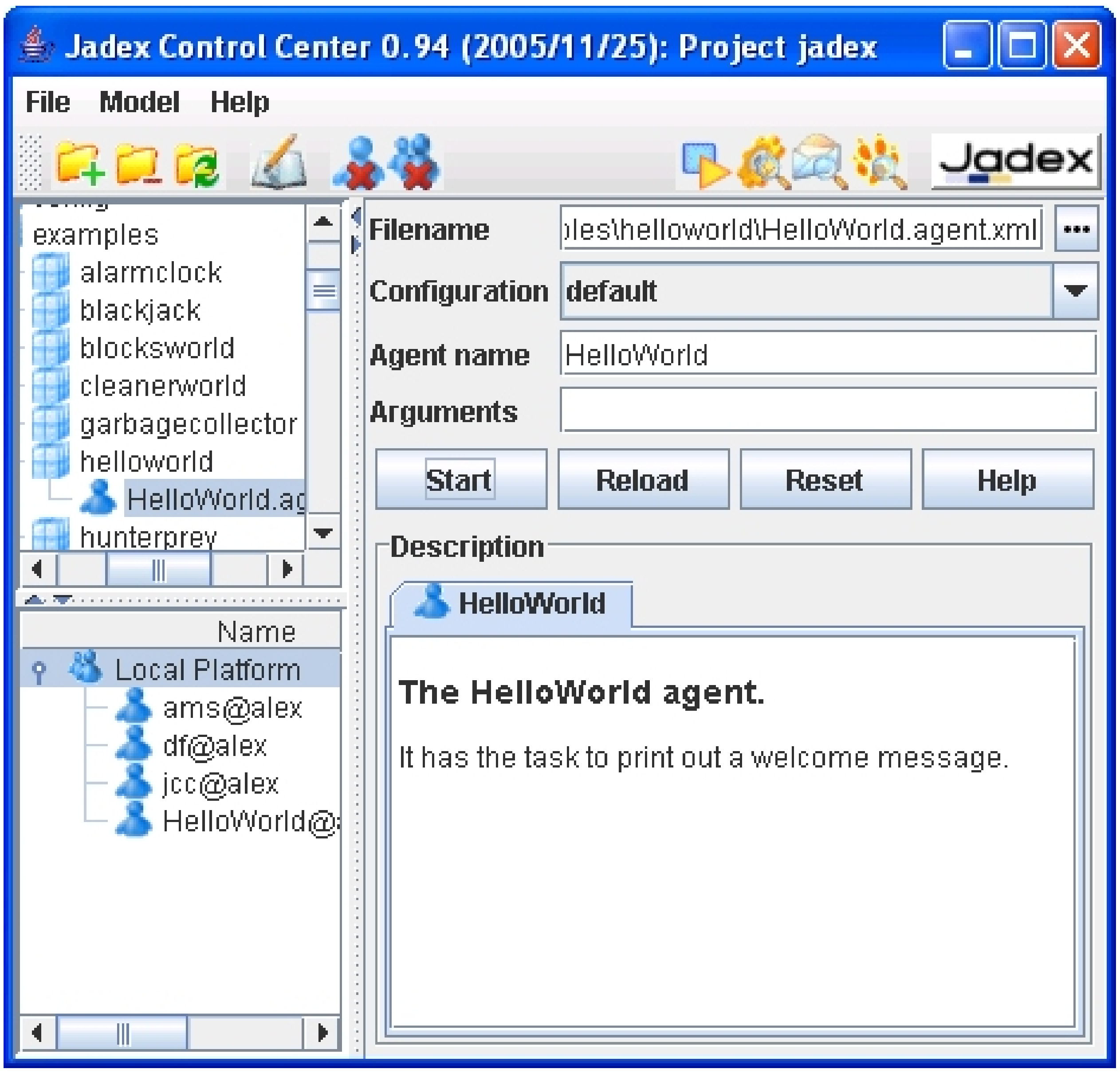
Note

The Control Center is realized as Jadex agent `jadex/tools/jcc/JCC.agent.xml` and is started per default when the Standalone platform is launched. To prevent the Control Center being started the `-nogui` option can be used for the Standalone platform. Using other adapters the Control Center can be launched by simply starting the corresponding JCC agent mentioned above.

2.1. Using the JCC

The "File" menu provides options for loading/saving the current project settings. In addition, the currently opened project name is displayed in the JCC window's title bar (see Figure 2.1, “JCC window”). The "File/Settings" menu item allows to open the platform settings dialog described in Section 2.2, “Platform Settings”. The "File/Exit" menu item allows to close the GUI and kill the JCC agent, and optionally shutdown the whole platform. The "Help" menu provides access to the Jadex help system and the Jadex homepage.

The buttons at the right side of the toolbar allow to switch between the perspectives provided by the plugins (starter, introspector, conversation center, and tracer). The buttons on the left as well as all menus except "File" and "Help" are dependent on the selected perspective and will be described there.



Filename: ›les\helloworld\HelloWorld.agent.xml

Configuration: default

Agent name: HelloWorld

Arguments:

The HelloWorld agent.

It has the task to print out a welcome message.

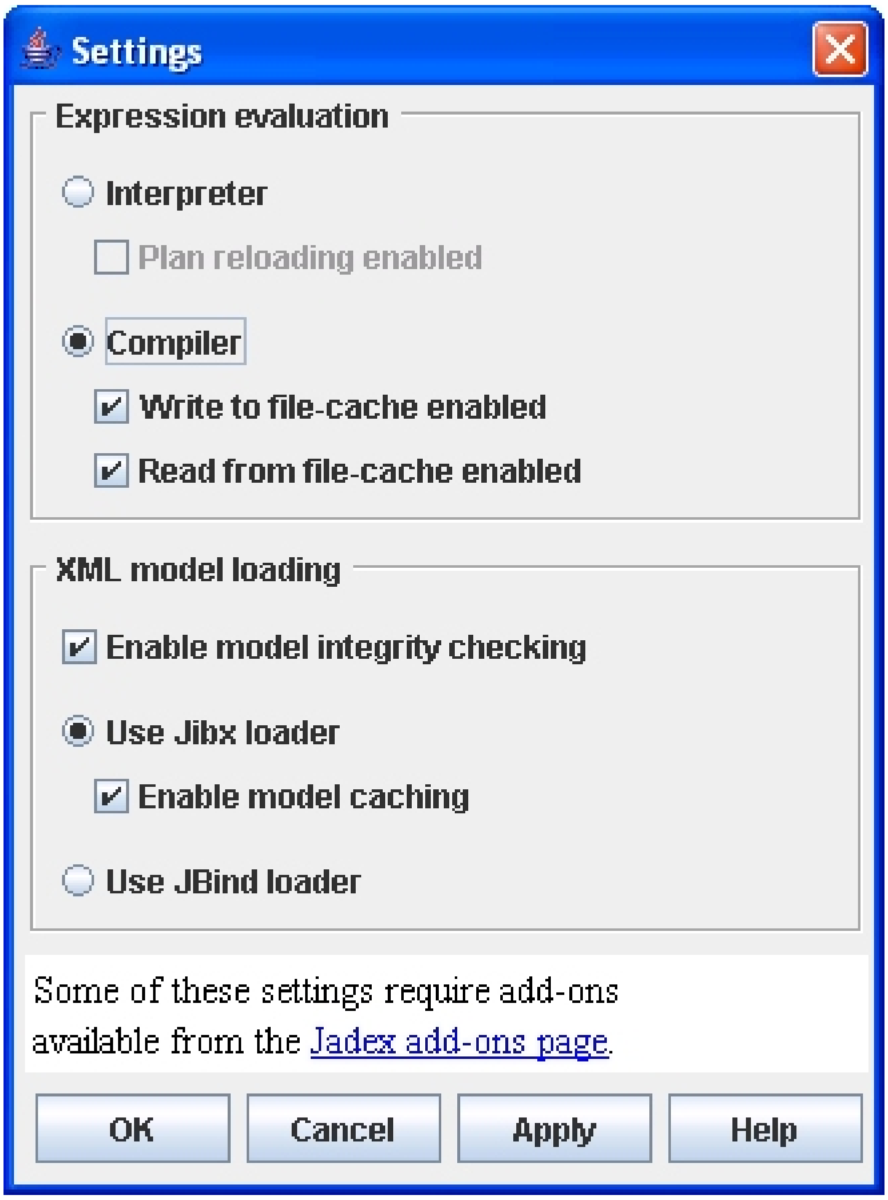
Figure 2.1. JCC window

2.2. Platform Settings

The settings dialog available from the "File" menu is used to set up diverse options for the Jadex platform and agent loading process. It is shown below in the Figure 2.2, “Platform settings dialog” and includes following options.

Expression evaluation. Jadex provides several options how to evaluate the Java expressions contained in ADFs. The built in interpreter features fast loading times, but limited runtime performance, as the Java statements have to be interpreted on every access. The interpreter allows to restart plan classes, when they have changed. This allows to try out a change in the Java code without having to reload the platform. Note, that this feature should only used for debugging, as it slows down the system. The alternative to the interpreter is the online compiler based on Janino, which is available from the Jadex homepage as an add-on. In order to further speed up the process of agent loading and execution, the compiled expressions may be saved directly to a cache file. The options Write to file-cache enabled and Read from file-cache enabled activate this behaviour.

XML model loading. The XML model loading section allows to influence how Jadex loads agent and capability models. When integrity checking is enabled all loaded agent and capability models are checked against a set of consistency rules (syntax of Java expressions, validity of cross references between goals and plans, etc.). The platform will refuse to start invalid agent models. The feature can be disabled to improve performance in deployed applications. To actually process the ADF and load the agent model, Jadex uses one of two XML-databinding frameworks. JiBX is a fast loader with a low memory footprint. To further improve performance the option to cache loaded models is available, which results in even faster loading times. JBind is a sophisticated mapping framework with the possibility to generate Java classes directly from the XML schema specification. Therefore JBind is the framework of choice for development purposes involving frequent changes to the XML schema.


Settings

Expression evaluation

- ○ Interpreter
  - ☐ Plan reloading enabled
- ● Compiler
  - ☑ Write to file-cache enabled
  - ☑ Read from file-cache enabled

XML model loading

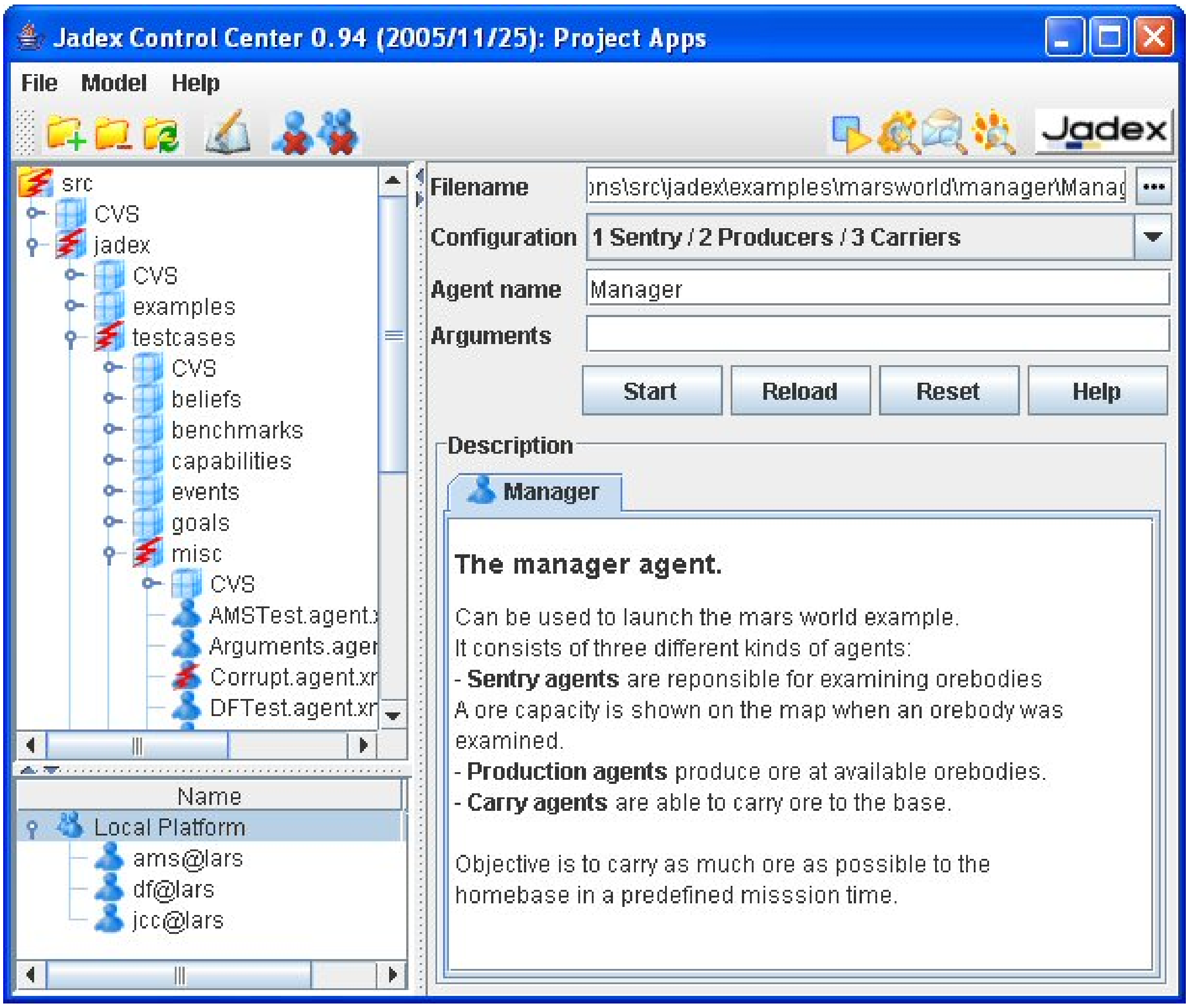
- ☑ Enable model integrity checking
- ● Use Jibx loader
  - ☑ Enable model caching
- ○ Use JBind loader

Some of these settings require add-ons available from the Jadex add-ons page.

Figure 2.2. Platform settings dialog

Chapter 3. Jadex Starter

The Starter is a central administration tool for managing Jadex agents. It offers basic functionalities for starting and stopping agents as well as more advanced ones for generating documentation and integrity checking of agent and capability models. In Figure 3.1, “Starter perspective” a screenshot of the Starter tool is depicted. The tool mainly consists of three different panels. On the upper left hand side the *Model Tree* is located. Below the Model Tree the *Running Agents* of the platform are shown. On the right hand side the *Model Panel* shows details of the currently selected agent or capability model.



The manager agent.

Can be used to launch the mars world example.
It consists of three different kinds of agents:
- **Sentry agents** are reponsible for examining orebodies
A ore capacity is shown on the map when an orebody was examined.
- **Production agents** produce ore at available orebodies.
- **Carry agents** are able to carry ore to the base.

Objective is to carry as much ore as possible to the homebase in a predefined misssion time.

Figure 3.1. Starter perspective

3.1. Model Tree

The model tree allows for easy navigation of Jadex agent and capability models. Each direct child of the root-node represent a *classpath* entry of the current project. Initially the model tree is empty.

- can be used to add a new directory to the model tree and to the classpath from which models are loaded.

- can be used to remove a directory from the model tree and the classpath.

Within the tree only agents and capability files are displayed according to the icons explained below:

- represents a Jadex agent model file. By selecting the file in the tree the model will be loaded and displayed in the Model Panel.
- represents a Jadex capability model file. By selecting the file in the tree the model will be loaded and displayed in the Model Panel.

View refresh. In addition to the standard start and stop functionalities the model tree also supports more advanced features. If not explicitly turned off in the `Model` menu the tree is automatically refreshed in certain time intervals. This means that changes on hard-disk are immediately reflected within the model tree. You can also initiate a refreshment directly by clicking the button.

Integrity checking. In addition all models found in the tree are automatically checked for integrity if this feature isn't turned off in the `Model` menu. This feature allows to effortlessly locate corrupt agent and capability files in the project. If a corrupt file is found, the file as well as all packages up to the root are marked as corrupt. A corrupt entity is displayed with a red bolt .

Jadexdoc generation. The Jadexdoc tool allows to generate documentation for agents in a style similar to Javadoc for Java classes. You can invoke the Jadexdoc generation dialog directly from the starter perspective via the button from the toolbar. If you previously have selected an agent or capability model or a package in the model tree this information will be passed to the dialog. The dialog will show this entry as "selected package". After the generation process the HTML output can be opened automatically in a browser. For details about the available options please refer to Chapter 8, *Jadexdoc Tool*.

3.2. Running Agents

The Running Agents panel shows all currently alive agents of the platform. For each agent its name and the first transport address is shown. To kill an agent it has first to be selected. Thereafter the kill action can be invoked via the popup menu or the toolbar .

Besides killing agents also the whole platform can be shutdowned via the button .

3.3. Model Panel

In the Model Panel details of a loaded agent or capability are shown. A model can be loaded either by selecting a file from the Model Tree or by using the "..." button to browse for a certain file. For a selected model several properties are presented:

- **Filename.** The exact filename of the displayed model.

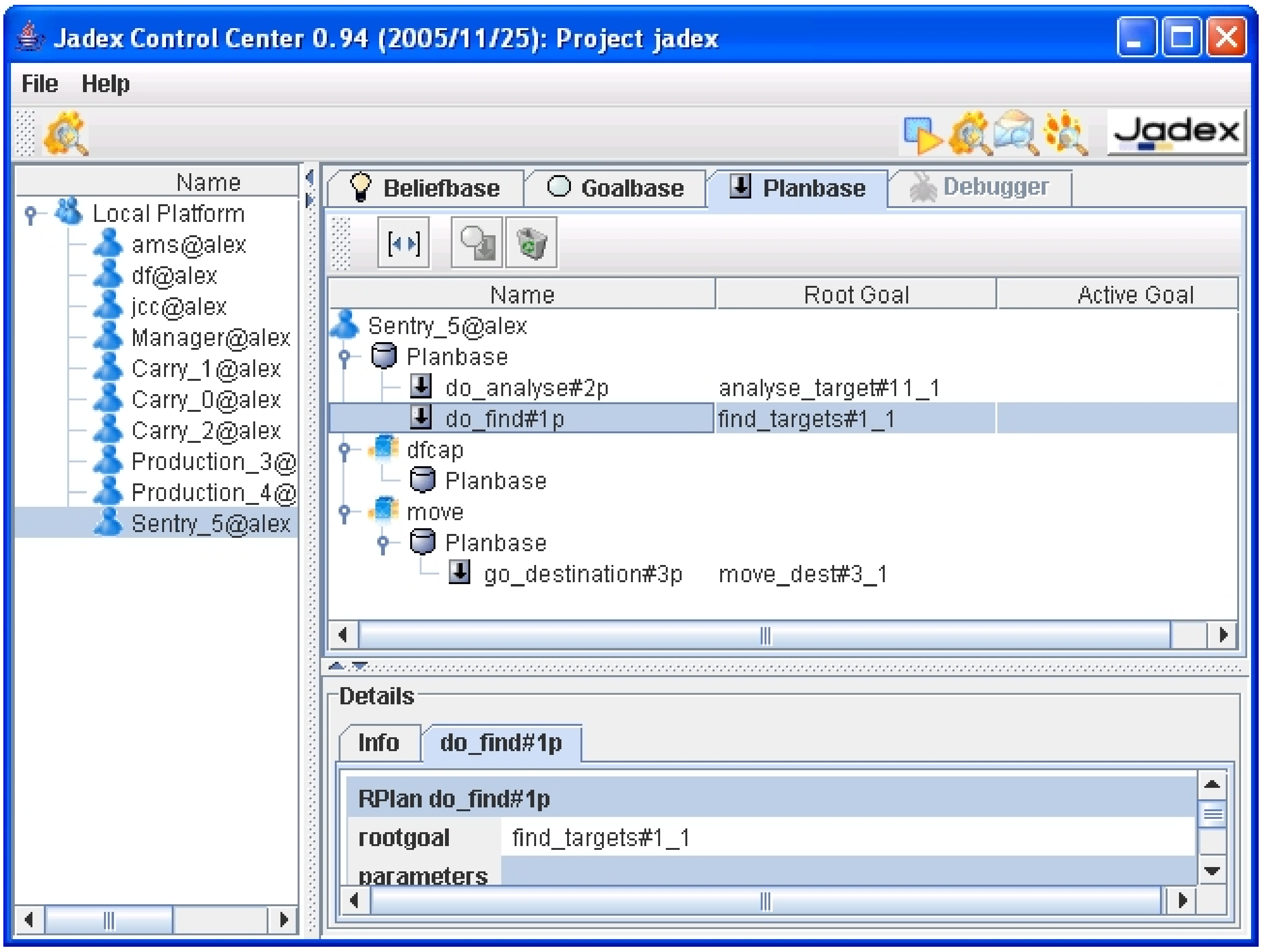
- **Configuration.** This choice contains all available configurations of the agent or capability. The default configuration of the agent or capability is selected.
- **Agent name.** The agent name is a necessary parameter for starting an agent. It represents the instance name for a new created agent from the loaded model. If you want to create more than one instance from a given model you need to change the instance name as agent names need to be unique.
- **Arguments.** The arguments are optional parameters for starting an agent. The arguments will be passed to the agent as `String`s. Several arguments are separated by spaces.
- **Description.** In the lower part of the Model Panel the description of the agent or capability is shown. The description is the HTML rendered output of the initial agent resp. capability comment of the model file. If the model contains errors an error report of all discovered bugs is displayed instead of the description.

If an agent model could be loaded without errors you can start a new agent instance of this model simply by hitting the Start button. If you changed a model you can load it from model again with the Reload button. The Reset button can be used to clear all fields and discard all loaded models from cache. Finally, the help button allows to invoke the online JavaHelp.

Chapter 4. Introspector

In the introspector perspective you can observe and manipulate the internal state of agents. In Figure 4.1, “Introspector overview” the introspector is shown while observing an agent from the marsworld example. You can use the agent tree at the left side to select agents you want to observe. The observation view for the selected agent is shown on the right side. In the observation views four different panels can be seen and chosen. The Beliefbase, Goalbase and Planbase tabs show the contents of the belief, goal and plan base, respectively. Alle these panels are described in Section 4.1, “Base Panels”. The Debugger tab allows to observe and control the event processing, consisting of plan selection and execution in the debugger panel.

Below the tabs in the observation window, a details panel shows details of elements (e.g. beliefs or events) selected with double click. This details panel is shared by all activated tabs, and therefore shows the last elements selected in any tab.



```
Name
Local Platform
    ams@alex
    df@alex
    jcc@alex
    Manager@alex
    Carry_1@alex
    Carry_0@alex
    Carry_2@alex
    Production_3@
    Production_4@
    Sentry_5@alex
```

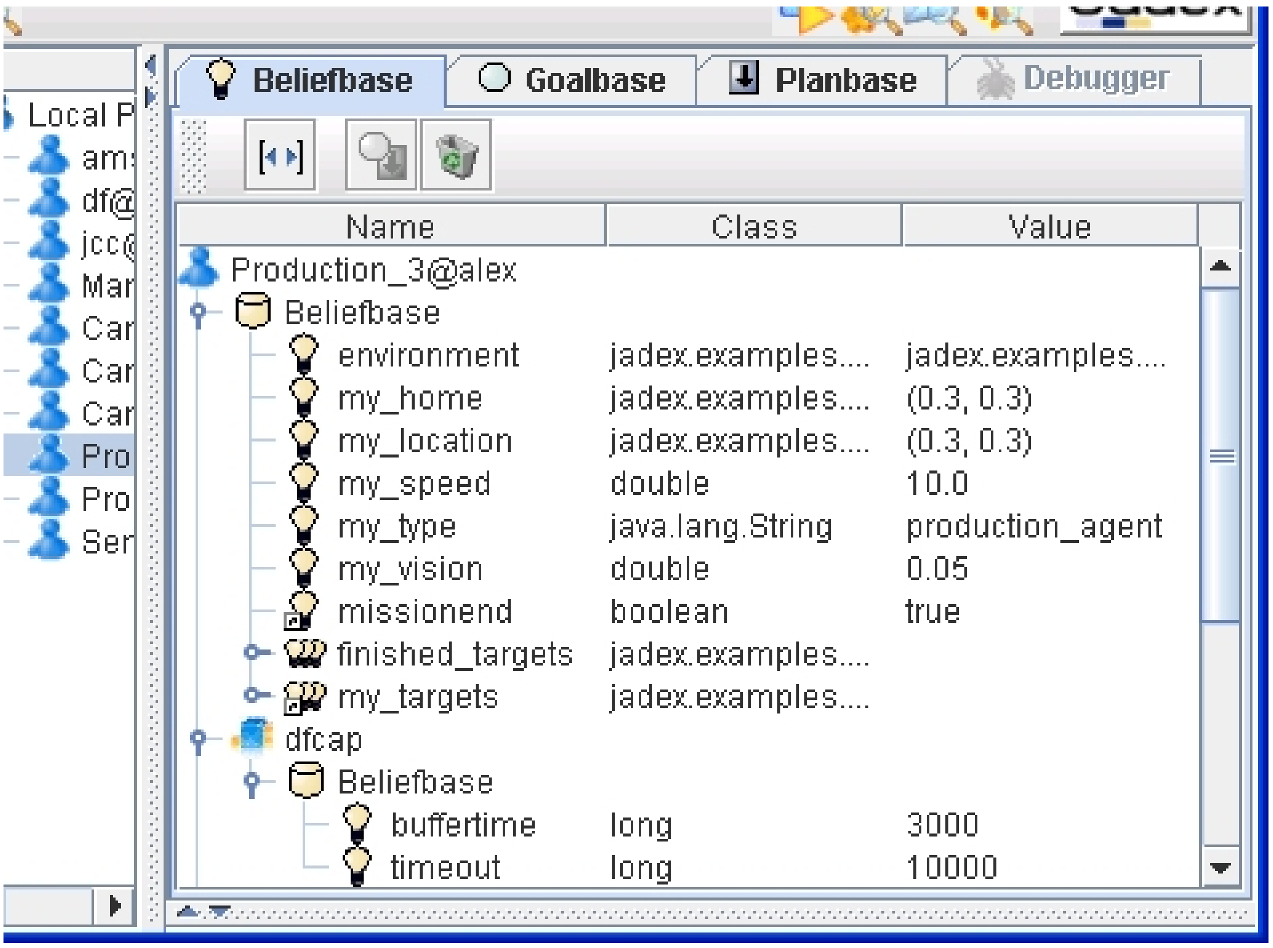
```
Name                          Root Goal               Active Goal
Sentry_5@alex
  Planbase
    do_analyse#2p             analyse_target#11_1
    do_find#1p                find_targets#1_1
  dfcap
    Planbase
  move
    Planbase
      go_destination#3p       move_dest#3_1
```

```
RPlan do_find#1p
rootgoal      find_targets#1_1
parameters
```

Figure 4.1. Introspector overview

4.1. Base Panels

Three base panels show the beliefs ( ), goals ( ), and plans ( ) respectively. They are very similar in their usage (see Figure 4.2, “The Beliefs Panel of Introspector”). All elements are shown in a tree structure representing the containment hierarchy of the elements in the capabilities of the agent.



Name	Class	Value
Production_3@alex		
Beliefbase		
environment	jadex.examples....	jadex.examples....
my_home	jadex.examples....	(0.3, 0.3)
my_location	jadex.examples....	(0.3, 0.3)
my_speed	double	10.0
my_type	java.lang.String	production_agent
my_vision	double	0.05
missionend	boolean	true
finished_targets	jadex.examples....	
my_targets	jadex.examples....	
dfcap		
Beliefbase		
buffertime	long	3000
timeout	long	10000

Figure 4.2. The Beliefs Panel of Introspector

The different elements of the agent are shown with different icons as explained in Table 4.1, “Introspector Base Panel Elements”. For each element the most important attributes are displayed directly in the tree/table structure. The content of the base panels will automatically be updated when changes occur inside the agent.

Table 4.1. Introspector Base Panel Elements

Element	Description
Agent	The agent containing all other elements
Capability	The capability with its contained elements
Belief Base	The beliefbase containing all beliefs of a capability
Belief	A single fact belief, or fact contained in a belief set
Belief Set	A belief set containing a number of facts
Referenced Belief	A belief visible in this capability, but declared elsewhere

Element	Description
Referenced Belief Set	A belief set visible in this capability, but declared elsewhere
Plan Base	The plan base containing all plans of a capability
Plan	A currently running plan
Goal Base	The goalbase containing all goals of a capability
Proprietary Goal	An adopted goal (active or inactive)
Process Goal	A goal currently being processed by some plan

The tree component allows nodes to be closed to focus on interesting subsets of the agent's functionality. The column widths can be (auto-) adjusted by dragging or double-clicking between the column headers. The table headers also provide a popup->menu (opened with right-click) that allows to hide some of the columns for better readability. You can double click on the elements to see more detailed information. The details panel is not automatically updated, you may have to double click the element again, to see up to date information.

You can to some extent manipulate the elements shown in the base panels. E.g. you can alter the values of facts in the beliefbase. Double click on the fact value you wish to change (in the value column), and then enter the new value as Java expression (i.e "text" for a string value). The expressions are evaluated using the imports as specified in the ADF (of the corresponding capability) therefore you can write expressions just as you would do that in a <fact> tag of the ADF. In addition, popup menus are available e.g. to remove beliefs, terminate plans or change the state of goals.

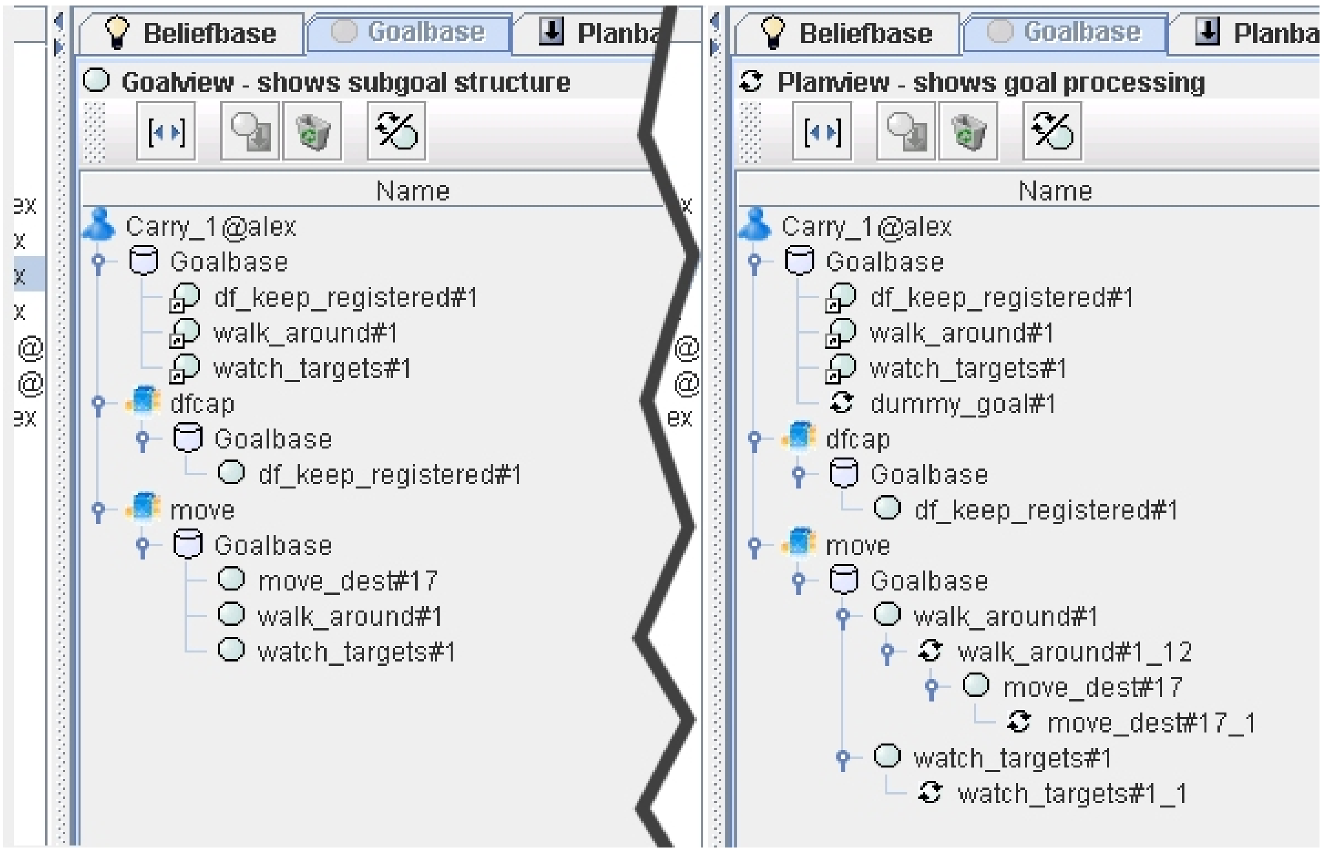
Some options are available to influence the appearance of the base panels (see Table 4.2, "Introspector Base Panel Options"). It is possible to deactivate (and later resume) the observation of any tab independently, by right-clicking on the tab's title. The observed agent will continue to execute, but changes will not be reflected in the panel until the observation is resumed. To pause the execution of the observed agent use the debugger panel described in the next section.

Table 4.2. Introspector Base Panel Options

[◂▸]	Resize all columns to fit to length of contained entries
○ ✗	Show / hide removed elements (e.g. finished plans)
🗑	Flush removed elements from memory
⟳○	Toggle goalview / planview (only for goalbase tab)

The resize icon allows to auto-adjust the widths of the table columns to fit the displayed content. The next two icons are concerned with the presentation of removed elements. Because the final state of a goal or a plan may be of interest after the goal has been dropped or the plan has terminated, the panels have an option not to remove those elements from the representation, but just visually mark them as removed. This is especially useful for debugging your agents, e.g. in conjunction with the debugger view described in the next section. To flush removed beliefs, goals and plans from the representation, you can use the waste bin icon.

The last icon is only available in the goalbase panel. It allows to switch between the goalview and the planview representation of the agent's current goals. The goalview, shown by default, displays only the proprietary goals (not the process goals created for each executed plan). The goals are arranged in the goal/subgoal hierarchy, i.e. a subgoal of a plan is shown as a subnode of the goal for which the plan is executed. Although this shows the goal hierarchy as expected, this is not how it is actually constructed inside the agent. The goalview hides that for each plan execution a process goal is created as a copy of the original goal. Subgoals dispatched by plans are actually created as subgoals of the process goal and not of the proprietary goal. For plans that are executed in reaction to internal or message events a dummy goal is created, which is also hidden in the goalview. Both views are compared in Figure 4.3, “Goalview and planview”. In the planview you can see that the `move_dest#17` goal was actually created by a plan as a subgoal of the `walk_around#1_12` process goal.



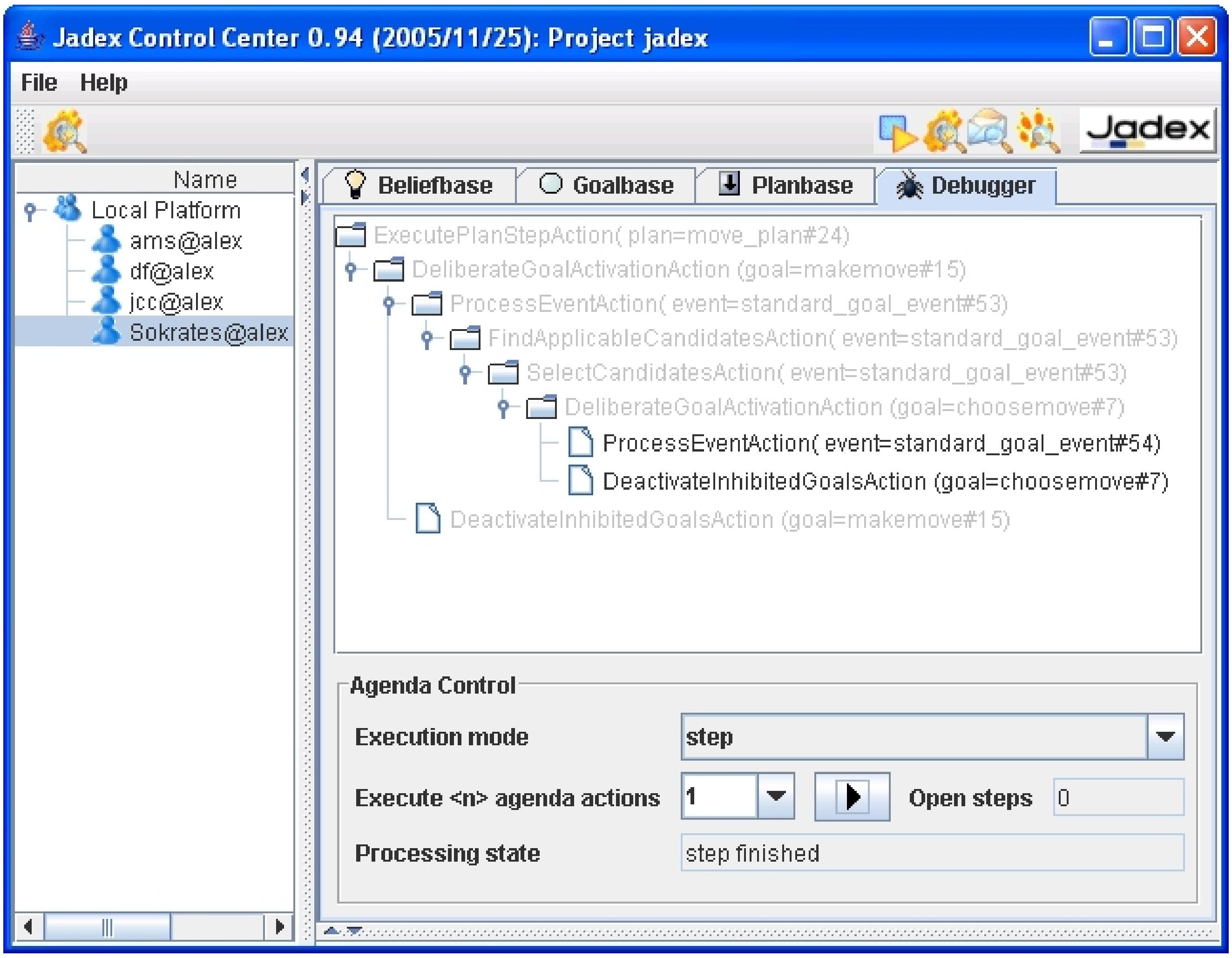
```
Goalview - shows subgoal structure
Name
Carry_1@alex
  Goalbase
    df_keep_registered#1
    walk_around#1
    watch_targets#1
  dfcap
    Goalbase
      df_keep_registered#1
  move
    Goalbase
      move_dest#17
      walk_around#1
      watch_targets#1
```

```
Planview - shows goal processing
Name
Carry_1@alex
  Goalbase
    df_keep_registered#1
    walk_around#1
    watch_targets#1
    dummy_goal#1
  dfcap
    Goalbase
      df_keep_registered#1
  move
    Goalbase
      walk_around#1
        walk_around#1_12
          move_dest#17
            move_dest#17_1
      watch_targets#1
        watch_targets#1_1
```

Figure 4.3. Goalview and planview

4.2. Debugger Panel

The debugger panel allows watching the event processing inside an agent. The window is made up of the agenda area, the agenda control area in the middle and the details view at bottom (see Figure 4.4, “The Beliefs Panel of Introspector”.


Jadex Control Center 0.94 (2005/11/25): Project jadex

```
ExecutePlanStepAction( plan=move_plan#24)
DeliberateGoalActivationAction (goal=makemove#15)
  ProcessEventAction( event=standard_goal_event#53)
    FindApplicableCandidatesAction( event=standard_goal_event#53)
      SelectCandidatesAction( event=standard_goal_event#53)
        DeliberateGoalActivationAction (goal=choosemove#7)
          ProcessEventAction( event=standard_goal_event#54)
          DeactivateInhibitedGoalsAction (goal=choosemove#7)
  DeactivateInhibitedGoalsAction (goal=makemove#15)
```

Agenda Control

Execution mode: step

Execute <n> agenda actions: 1

Open steps: 0

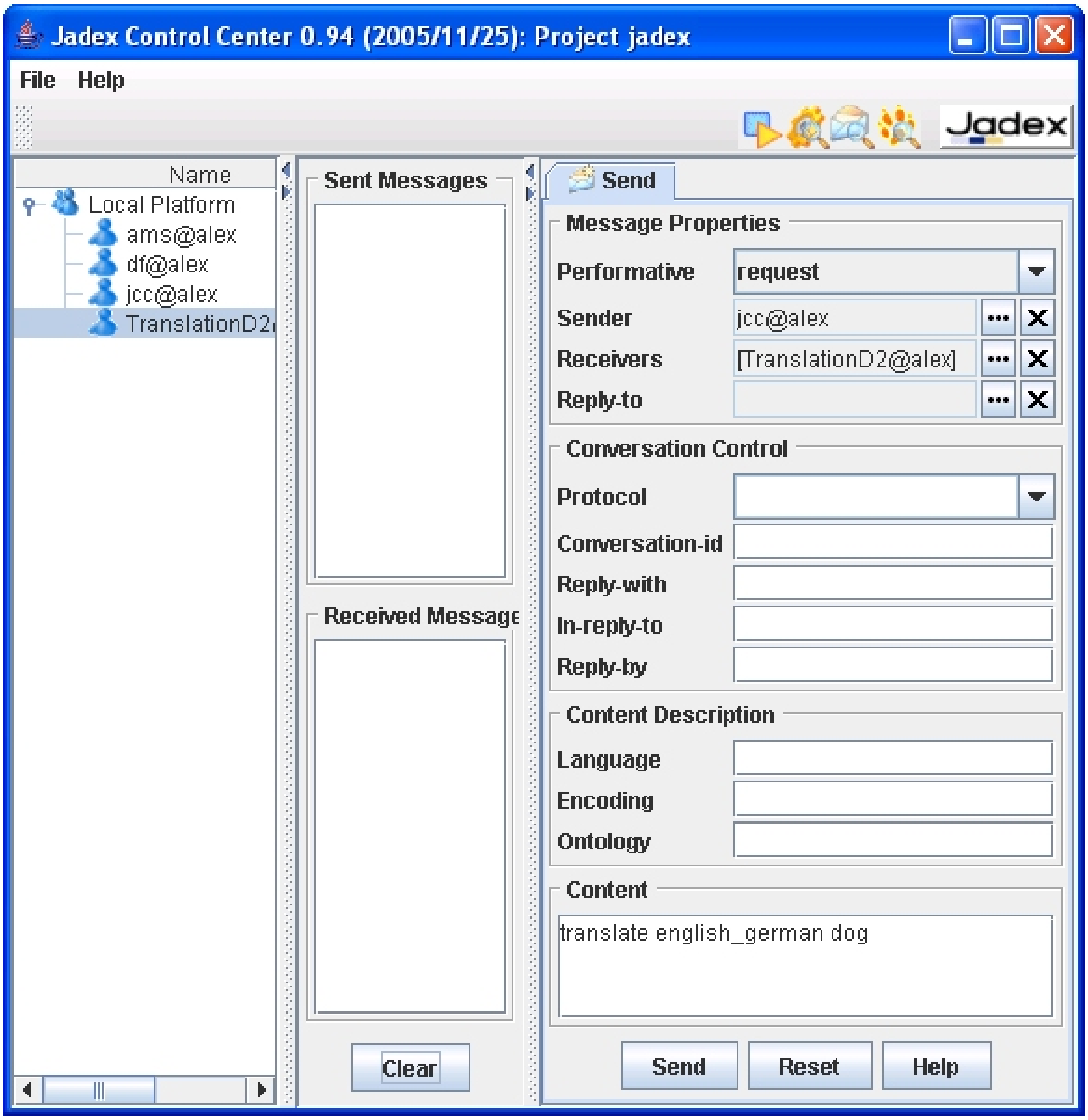
Processing state: step finished

Figure 4.4. The Beliefs Panel of Introspector

The agenda contains all current action entries of the agent, whereby entries in light-grey denote already processed actions. In addition it can be seen whether entries currently have a valid precondition and thus can subsequently be executed; otherwise they are marked as (invalid). Such invalid actions will not be executed, but just ignored by the execution mechanism. To see some more information about an agenda action you can click on it and inspect its values in the details view. In addition to observing the agent's internal behaviour the tool also allows you to control the agenda execution by performing actions in step mode. If the execution mode is set to step or cycle you can use the forward button to execute as many steps as shown in the execute <n> agenda actions choice. The difference between step and cycle mode concerns only the execution of ProcessEventActions that are decomposed to finer-grained sub steps (FindApplicableCandidatesAction, SelectCandidatesAction and ScheduleCandidatesAction) when the step mode is activated. Therefore, the step mode allows you to examine the details of the BDI plan selection process what can be helpful in understanding and explaining unexpected application behaviour. The open steps status bar shows the progress of the action execution by highlighting the number of steps the agent still has to perform whereas in the the processing state line it can be seen if a step is currently requested or has been finished.

Chapter 5. Conversation Center

The conversation center can be used to compose messages, send messages to agents and inspect the received answers. The left part of the panel (see Figure 5.1, “Conversation center overview”) contains two lists for the latest sent and received messages. Double clicking on a message from one of the lists will show the message contents as a new tab on the right side.



Performative: request
Sender: jcc@alex
Receivers: [TranslationD2@alex]
Reply-to:
Protocol:
Conversation-id:
Reply-with:
In-reply-to:
Reply-by:
Language:
Encoding:
Ontology:
Content: translate english_german dog

Figure 5.1. Conversation center overview

5.1. Sending Messages

The send tab will always be present and allows to compose a new message for sending. The message format follows the FIPA standards (see http://www.fipa.org). You can choose an appropriate performative for your message by using the drop down list of available performatives. The sender is by default initialized with the name of the control center agent. The receivers specify which agents will receive the message. If answers to the message should not be sent to the original sender (default) you can provide an optional reply-to agent. The agent identifiers for sender, receivers, and reply-to are selected using a separate dialog, which can be accessed by clicking the "..." button besides the text field. To clear the agent identifiers click on the "x" button to the right of the "..." button.

The other attributes of the message can be entered as plain strings. For the protocol you can also select one of the protocol types predefined by FIPA. Normally, only a few slots need to be filled in for a message. See e.g. Figure 5.1, “Conversation center overview”, which shows a message commonly used in the [Jadex Tutorial]. After composing the message it can be sent simply by hitting the "Send" button. If it was successfully sent, a copy of the message will be placed in the sent messages list.

You can later reopen the message in a new tab by double-clicking it in the list. To resend the message without changes click the "Send again" button in the newly opened tab. If you wish to change the message before re-sending it use the "Edit" button instead, which will fill in the slots of the message in the send tab, so you can edit it.

5.1.1. Agent Selector Dialog

The left list of the agent selector dialog (see Figure 5.2, “Agent selector dialog”) shows the agents currently running on the platform. You can double-click on an agent from the list to select it. The upper list on the right side shows the currently selected agent(s). The buttons below the list allow to add a new agent identifier from scratch, which can be edited below the list. When you finished selecting agents, hit the "Ok" (or "Cancel") button at the bottom of the dialog.

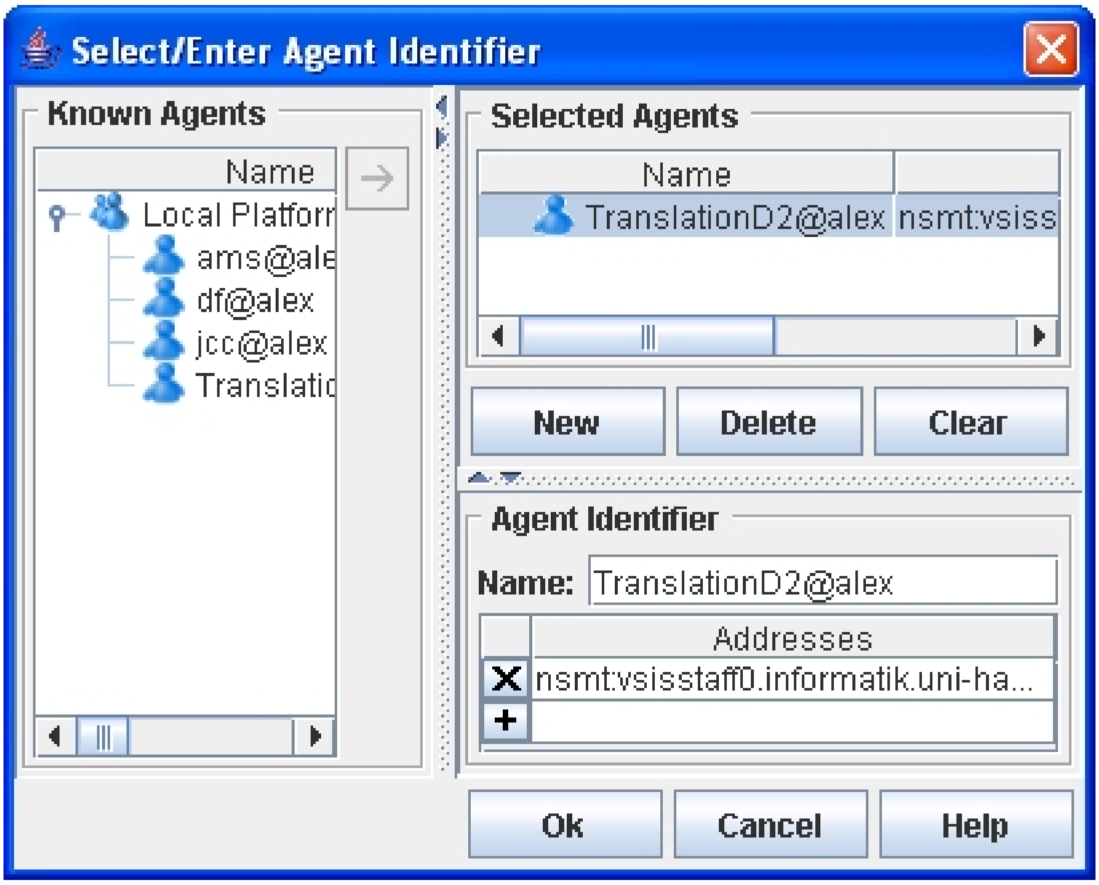
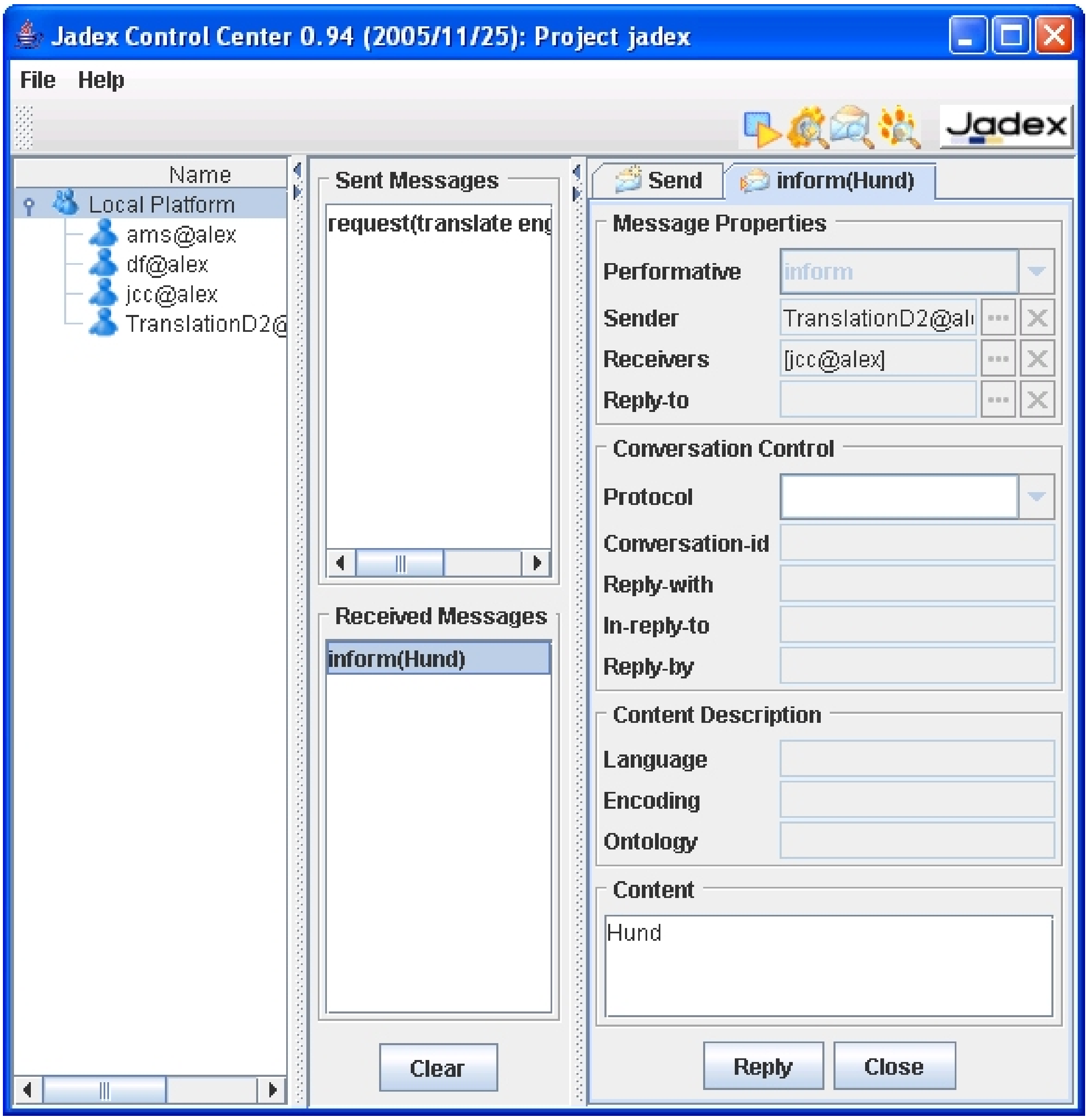


Figure 5.2. Agent selector dialog

5.2. Receiving Messages

The messages received by the conversation center agent are placed in the received messages list. Double-clicking on a received message will open it as a new tab (see Figure 5.3, “A received message”). When you want to reply to the message, click on the "Reply" button. The user interface will switch to the send panel, and fill in all slots (receiver, conversation-id, etc.) based on the received message. You can then edit the reply message and hit the "Send" button. If you want to get rid of the tab of a received message, you can use the "Close" button next to the "Reply" button.



request(translate eng

inform(Hund)

Performative: inform
Sender: TranslationD2@al
Receivers: [jcc@alex]
Reply-to:
Protocol:
Conversation-id:
Reply-with:
In-reply-to:
Reply-by:
Language:
Encoding:
Ontology:
Content: Hund

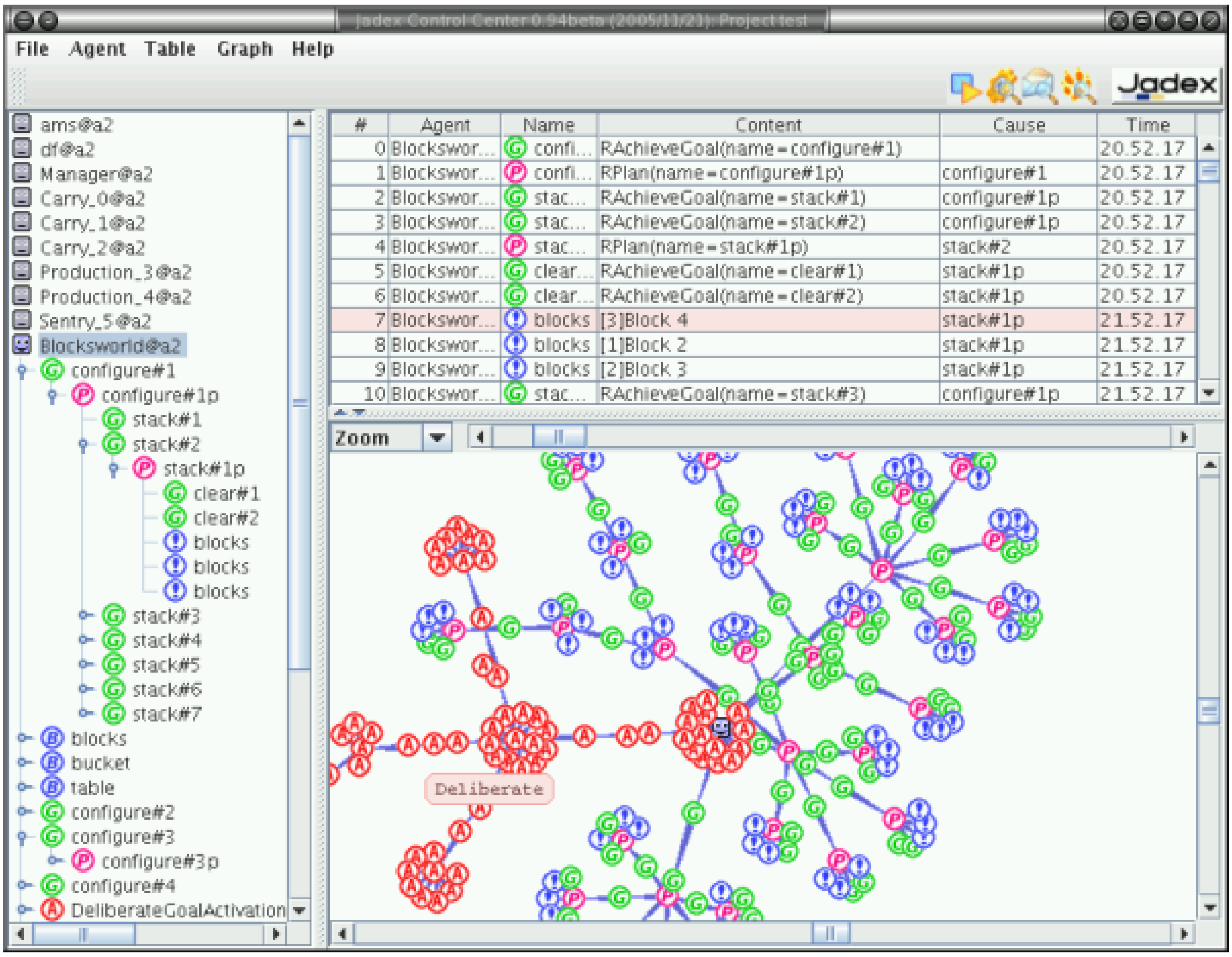
Figure 5.3. A received message

Chapter 6. BDI Tracer

The tracer is accessible from the Jadex Control Center tool menu. It is a tool inspired by the Ph.D. work of Dung N. Lam working on agent software comprehension with abductive reasoning. The tracer provides basically an interface and means to log the internal state of a BDI agent, and to analyze and visualize the logged information. It is made of two components. The first one is the TracerAdapter placed in front of an agent as a tool adapter. It is responsible for filtering messages concerning the tracing process away from the message queue of an agent and it collects the information about agent's internal state changes and other occurrences in the system. The information is then sent to the Tracer Agent, if an instance is present on the platform. The latter has the duty to analyze the traces, store them, and to present them to the user in a graphical form.

6.1. Main Window

The main window of the tracer agent may be seen in Figure 6.1, “Tracer Main Window”. The traces presented here are from the Blocksworld example. In order to indicate a type of traces, they are marked with icons and their different meaning is explained in Table 6.1, “Information logged by the tracer”. The tracer perspective is split into three views including a tree view of agents and traces, a tabular view of traces and a 2D trace space exploring panel.



#	Agent	Name	Content	Cause	Time
0	Blockswor...	confi...	RAchieveGoal(name=configure#1)		20.52.17
1	Blockswor...	confi...	RPlan(name=configure#1p)	configure#1	20.52.17
2	Blockswor...	stac...	RAchieveGoal(name=stack#1)	configure#1p	20.52.17
3	Blockswor...	stac...	RAchieveGoal(name=stack#2)	configure#1p	20.52.17
4	Blockswor...	stac...	RPlan(name=stack#1p)	stack#2	20.52.17
5	Blockswor...	clear...	RAchieveGoal(name=clear#1)	stack#1p	20.52.17
6	Blockswor...	clear...	RAchieveGoal(name=clear#2)	stack#1p	20.52.17
7	Blockswor...	blocks	[3]Block 4	stack#1p	21.52.17
8	Blockswor...	blocks	[1]Block 2	stack#1p	21.52.17
9	Blockswor...	blocks	[2]Block 3	stack#1p	21.52.17
10	Blockswor...	stac...	RAchieveGoal(name=stack#3)	configure#1p	21.52.17

Figure 6.1. Tracer Main Window

Table 6.1. Information logged by the tracer

Ⓐ	Actions describe internal processes of an Agent and are by default ignored by the tracer.
Ⓑ	Beliefs are meta traces, that collect all traces concerning the use of beliefs and their change.
❗ ❓	Belief read and write access indicate that an agent or its plans have accessed a belief and possibly changed it.
↘ ↗	The icons stand for message receive or send events respectively.
Ⓖ	Goals are traced when they are adopted by the agent.
Ⓟ	Plans are shown, when created in response to a goal or event.
Ⓔ	Events represent any agent state changes.

6.1.1. Trace Tree

At the left there is a tree view showing all Jadex BDI agents known to the tracer. The meaning of a particular icon depicting an agent is shown in Table 6.2, “State of an agent”. Descending from the BDI agents, all traces are linked beneath nodes identified to be their cause.

Table 6.2. State of an agent

😐	The agent did not send any traces yet.
🙂	The agent sends its traces to the tracer.
😑	The agent is ignored by the tracer.
💀	The agent died.

The functionality provided by the tree popup menu is similar to the functionality from the Agent Menu (Section 6.2.1, “Agent Menu”) and concerns the currently selected trace or agent. In the case of an agent the user may choose to observer it and to adjust the trace filter and history limit (Filter). The other options allow to show or hide traces in the graph or table. With the last menu item the trace or the agent may be removed from the tracer perspective.

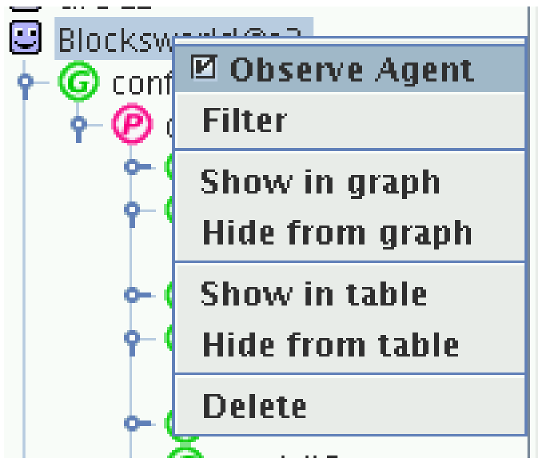


Figure 6.2. Tree Popup Menu

6.1.2. Trace Table

On the right there are two views. The upper one shows a tabular view of the traces. The traces are ordered in sequence they arrive. The table shows information like a unique trace id. For plans and goals it is the instnace name. For beliefs it is the name of the belief. Also shown is the value, causes and the time this trace happened. The value show the content of a specific trace. For beliefs it is the value of the beliefs. For plans and goals, the string representation of the runtime instance.

The user may select traces in the table based on different criteria, remove them from the table, show them in the graph panel or delete from the tracer perspective. All the fuctionality is accessible under the Table Menu (Section 6.2.2, “Table Menu”) and a coresponding popup menu.

6.1.3. Trace Exploration Graph

The lower of the two right views is the graph panel allowing to explore the space of traces. It also allows to multiple presentation options, navigation and choice of among the traces. All this functionality may be accessed from the Graph Menu (Section 6.2.3, “Graph Menu”) and a graph popup menu (Figure 6.5, “Graph Popup Menu”).

6.2. Menus

The menu provides access to functions concerning the tracer agent itself and the BDI agents analyzed. Functions corresponding to the tabular view and the 2D graph view are also accessible from here.

6.2.1. Agent Menu

Under this menu (cf. Figure 6.3, “Agent Menu” concerning agents the user has the option to:

- Observe - an agent. This will tell the agent to send its traces to the tracer.
- Observe all - will cause all BDI agents (known to the tracer) to send their data.
- Ignore - an agent. Has the complementary effect to Observe.

- Ignore all - is the reverse of Observe all.
- Ignore at first - causes the tracer to ignore all newly occurring agents.
- Show in graph - tells the tracer to show the traces of an agent in the 2D graph as soon as they arrive.
- Hide from graph - removes all agent traces from the 2D graph.
- Show in table - tells the tracer to show the traces of an agent in the table.
- Hide from table - removes all agent traces from the table.
- Delete - removes the agent and corresponding traces from all views.
- Delete dead agents - removes all dead agents and their traces from all views.
- Filter - shows a filter dialog for the current selected agent.
- Default filter - shows a filter dialog for a prototypical agent all new agents will inherit their properties from.

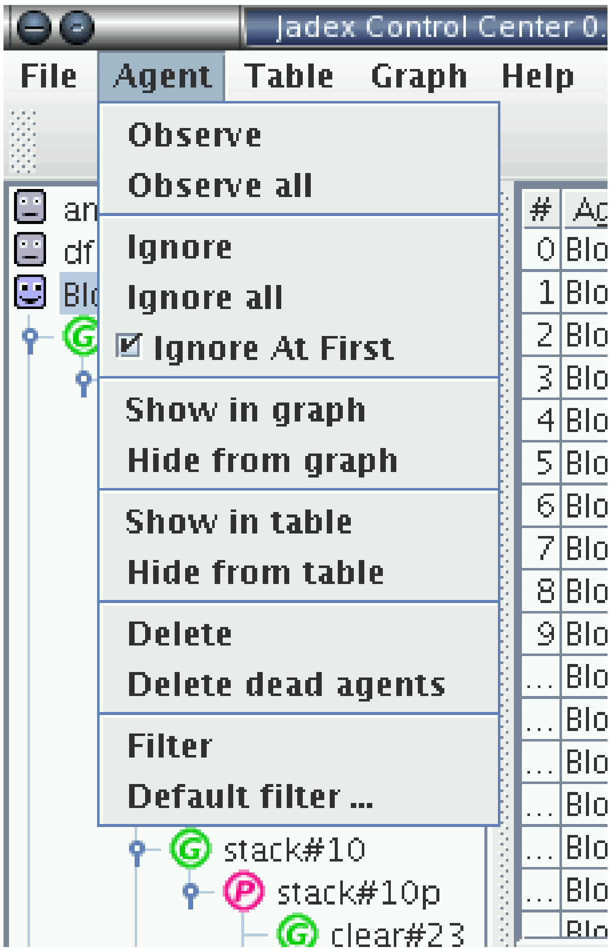


Figure 6.3. Agent Menu

6.2.2. Table Menu

The Table menu (cf. Figure 6.4, “Table Menu”) provides functionality concerning the table view. Following options are available to the user:

- Select causes - selects immediate causes of selected traces.
- Select effects - selects the immediate effects.
- Show in graph - shows all selected traces in the 2D graph.
- Hide in graph - hides selected traces from the 2D graph.
- Remove - removes traces from the table.
- Delete - deletes the traces and removes them from all views.
- Scroll - tells if the table should be scrolled, when new traces arrive.

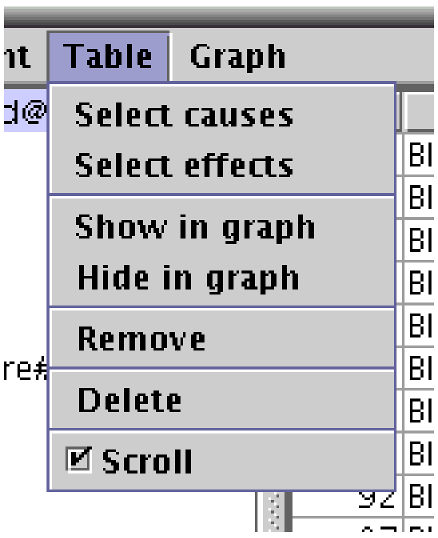


Figure 6.4. Table Menu

6.2.3. Graph Menu

The tracer graph menu is accessible from the main menu and as pop-up in the graph view(cf. Figure 6.5, “Graph Popup Menu”). It provides access to following functions:

- Show - is used to show actions, beliefs or messages connected to trace nodes already shown in the graph. If a trace is selected, the user may choose to show the causes and effects of that trace.
- Hide - hides actions, beliefs or messages from the graph view. A single trace, its causes or its effects may be removed form the view.
- Expand - will expand the trace by one level of the causes or effects.
- Collapse - will shrink and hide the traces around the selected one.
- Delete - will remove the trace from all views in the tracer.
- Join Beliefs - may be used to collapse all belief access nodes into a single one.
- Join Messages - will join the send and receive events of a message with an edge, therefore establishing connections between agents.

- Labels - this check-box indicates that the traces in the graph should be shown with thier corresponding labels instead of an anonymous icon. The labels ar turcated by length and a selected trace is always shown with its label.

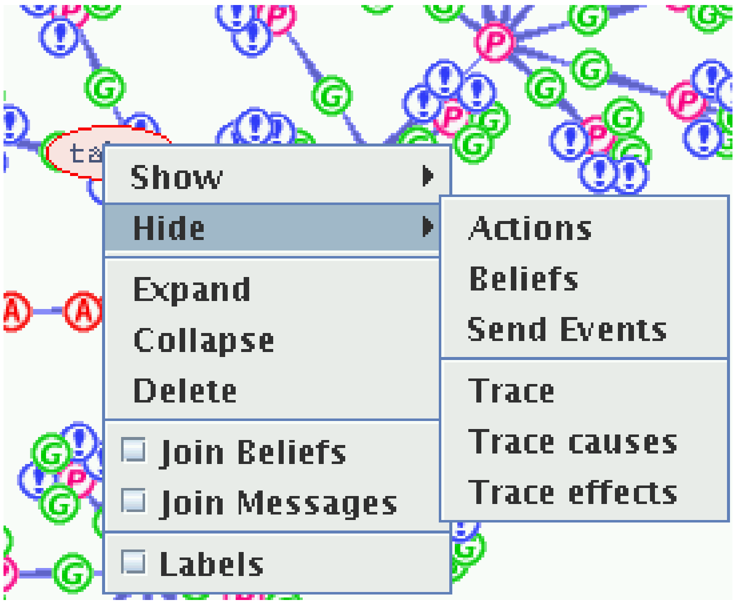
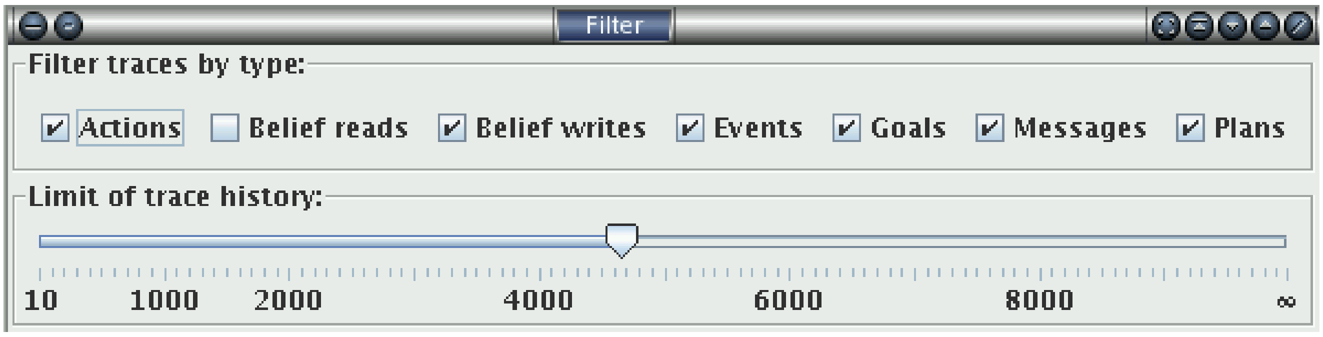


Figure 6.5. Graph Popup Menu

6.3. Agent Filter Dialog



Filter traces by type:

[✓] Actions [ ] Belief reads [✓] Belief writes [✓] Events [✓] Goals [✓] Messages [✓] Plans

Limit of trace history:

10 1000 2000 4000 6000 8000 ∞

Figure 6.6. Agent Filter Dialog

The filter dialog can be accessed from the Agent Menu or from the agent popup menu (see Figure 6.6, “Agent Filter Dialog”). There is also a Default filter menu item allowing to set the same values for a prototypic agent used for new agents introduced to the tracer.

It allows the user to customize, what kind of traces are interesting for her. The trace type filter can be used to choose among different kinds of traces. The fillter will pass all traces by kind if the corresponding check box is marked. The limit of trace history below allows to store only a specified number of traces in the tracer and may be set for each agent. Moving the slider to the right-most position implies that no limit will be enforced.

Changes in the dialog are applied immeadiately. The dialog will hide when it looses the focus.

Chapter 7. Beanynizer

The Beanynizer is a plugin for the widely used ontology development environment Protégé™ and allows to generate JavaBeans and a JADE™ ontology file from a modelled ontology. It is very similar to the well known beangenerator plugin™ by Acklin but offers some more flexibility regarding the generated code.

7.1. Installation

The description of the installation process assumes that you have successfully downloaded and installed Protégé™ 2.1 (or later). The installation of the plugin is simple. Extract the `plugin.zip` file from the beanynizer distribution into the `protege/plugins` directory. Make sure to use the “Use folder names” option (or similar) of your zip tool, such that a subdirectory `protege/plugins/jadex.tools.beanynizer` is automatically created. (You can also create this directory by hand before unzipping. In the end you should have a `jadex_beanynizer.jar`, a `plugin.properties` file, and some additional jar files in this directory.) Now you can start Protégé™ as usual. The Help → Plugins menu should contain an entry Jadex Beanynizer. Selecting this entry will just take you to the Jadex homepage. For the usage of the plugin see the next sections.

7.2. Creating an Ontology

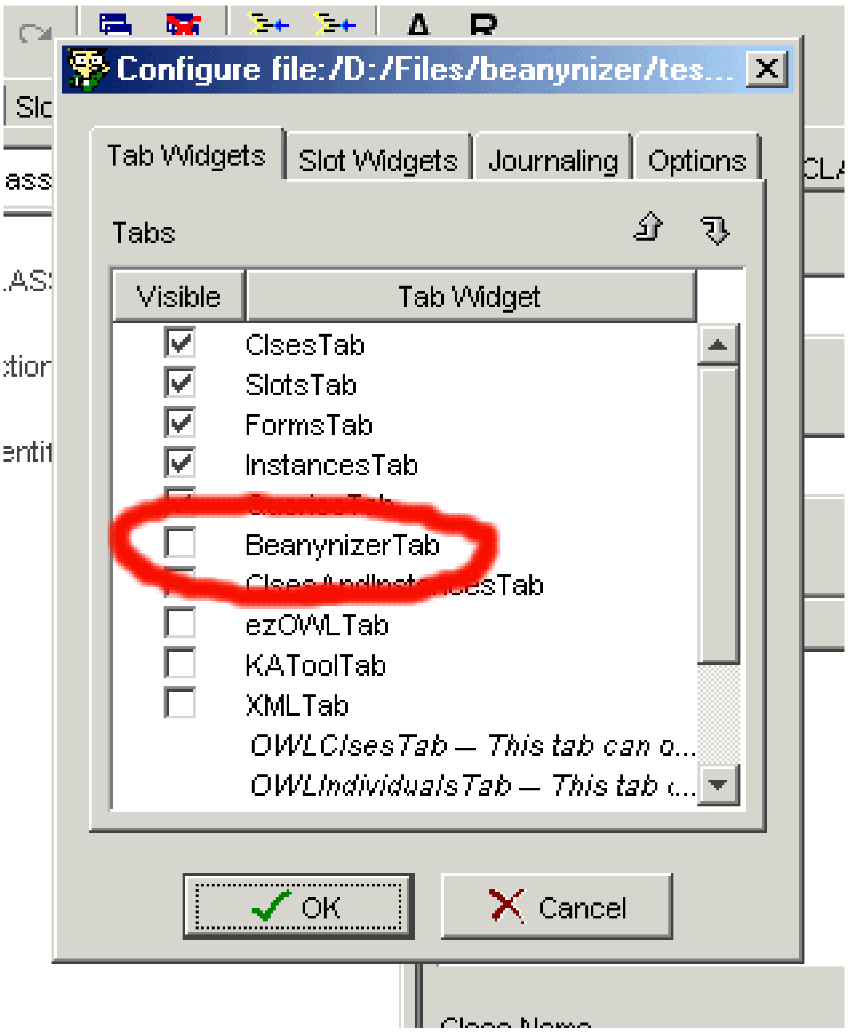
This section only discusses details of the usage of the Beanynizer plugin. For general information about creating ontologies in Protégé™ please consult the Protégé™ documentation. To create an ontology for use with Jadex, follow these four steps:

- Create a new ontology e.g. with the Project → New... menu item. Note that Beanynizer currently does not support OWL, so you have to choose a standard or RDF ontology format. Save the new ontology to a directory of your choice.

- Include one of the Beanynizer default ontologies. The beanynizer supports the creation of ontologies for use with the JADE platform (`beanynizer_default.pprj`), or pure Java ontologies for use with the Jadex Java-XML encoding (`beanynizer_beans_default.pprj` or `beanynizer_beans_fipa_default.pprj` if you want to refer to FIPA related concepts). E.g., Use the Project → Include Project... menu and select the `beanynizer_beans_fipa_default.pprj` file in the appearing file chooser. Protégé™ will store the location of the default ontology using an absolute path. As this is undesirable most of the time, you should copy the Beanynizer default files (`.pprj`, `.pins`, `.pont`) to the directory of your ontology, and include the ontology from there. In this case Protégé™ will use a relative path name.

- Add classes and slots to your ontology. The JADE default ontology provides four base classes (`Concept`, `AgentAction`, `agent-identifier`, `Predicate`) that you should use as superclasses for your own concepts. If you don't know the meaning of thesebase classes consult the JADE™ ontology guide. The for a pure Java ontology, classes can be directly created as subclasses of Protégé's :THING class. In any case, you may also find it helpful to take a look at the ontologies used in the Jadex examples.

- Generate Java sources from the ontology using the Beanynizer tab. If you created your ontology from scratch, you will have to activate the tab first. Select the Project → Configure... menu and open the Tab Widgets tab (see Figure 7.1, “Protégé™ plugin configuration”). Activate the Beanynizer tab and close the dialog by hitting Ok. In the Beanynizer Tab you can now edit the code generation options such as packagename and output directory (see Figure 7.2, “Jadex Beanynizer tab”). Depending on the base ontology you used, you also have to select the correct Generation Mode (Java for a pure beans ontology, Jade for a JADE ontology). Pressing the Generate Files button will create the desired source files. See the Jadex user guide, for an introduction how to use the generated ontology in your Jadex agents. The next sections discuss how you can influence the code generationprocess.



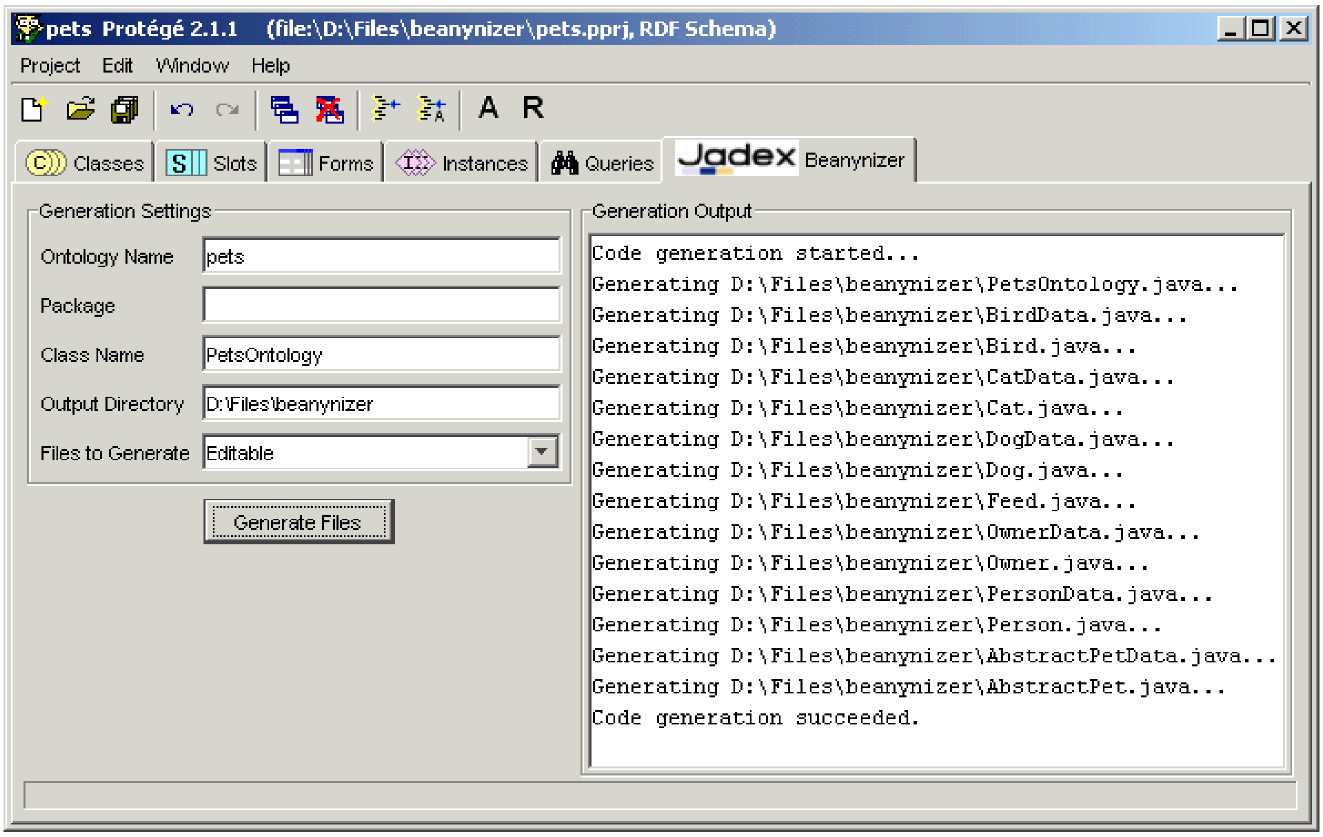
Configure file:/D:/Files/beanynizer/tes...

Visible	Tab Widget
✓	ClsesTab
✓	SlotsTab
✓	FormsTab
✓	InstancesTab
✓	[illegible]
☐	BeanynizerTab
✓	[illegible]esTab
☐	ezOWLTab
☐	KATool Tab
☐	XMLTab
	OWLClsesTab — This tab can o...
	OWLIndividualsTab — This tab ...

Figure 7.1. Protégé™ plugin configuration

7.3. Ontology Options

The general options for the ontology are available from the code generation panel (see Figure 7.2, “Jadex Beanynizer tab”). The ontology name is the name, that will appear in the ontology slot of an ACL messages. From Java this is available with the *`ontologyclass`*`.ONTOLOGY_NAME` constant. A package can be specified, where the ontology class file should be generated. This package is also the default for other generated classes. The class name is the Javaclass name to be used for the ontology (without package). The output directory is the root directory for the package hierarchy to be generated. You can use relative paths here, which will be expanded relative to the saving-location of the Protégé™ project. When subdirectories for some packages do not exist, they will be created on-the-fly.



Ontology Name	pets
Package	
Class Name	PetsOntology
Output Directory	D:\Files\beanynizer
Files to Generate	Editable

```
Code generation started...
Generating D:\Files\beanynizer\PetsOntology.java...
Generating D:\Files\beanynizer\BirdData.java...
Generating D:\Files\beanynizer\Bird.java...
Generating D:\Files\beanynizer\CatData.java...
Generating D:\Files\beanynizer\Cat.java...
Generating D:\Files\beanynizer\DogData.java...
Generating D:\Files\beanynizer\Dog.java...
Generating D:\Files\beanynizer\Feed.java...
Generating D:\Files\beanynizer\OwnerData.java...
Generating D:\Files\beanynizer\Owner.java...
Generating D:\Files\beanynizer\PersonData.java...
Generating D:\Files\beanynizer\Person.java...
Generating D:\Files\beanynizer\AbstractPetData.java...
Generating D:\Files\beanynizer\AbstractPet.java...
Code generation succeeded.
```

Figure 7.2. Jadex Beanynizer tab

The Files to Generate option specifies which kind of Java files should be generated for your ontology classes. Note that this option only represents a default, that can be overridden individually for each ontology class as described in the next section.

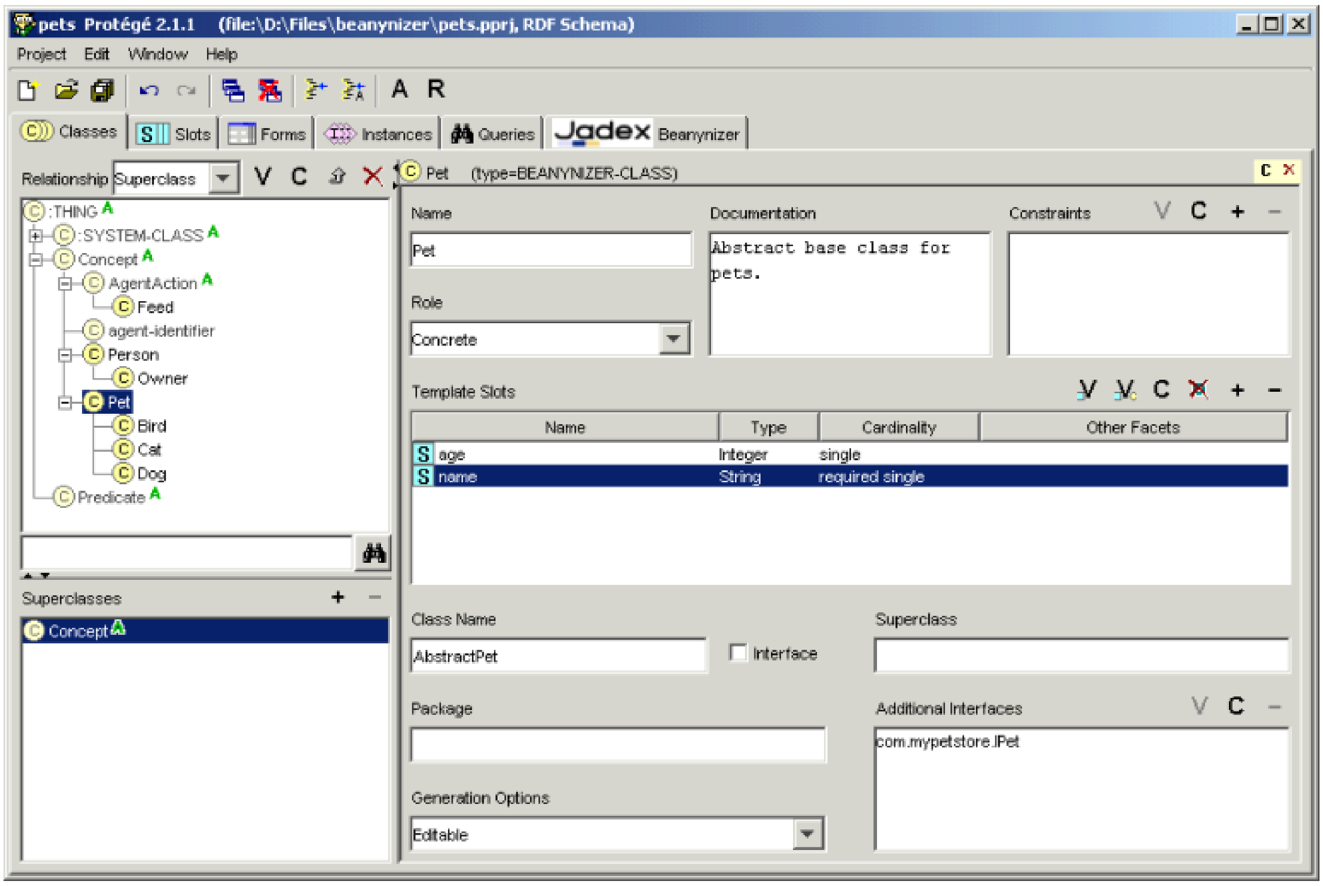
External
: means that the ontology uses Java classes that already exist and do not have to be generated. In this case, only the single ontology class file will be generated.

Editable
: (which is the default) creates two files for any ontologyclass: A `*classname*Data.java` file, which contains the required fields and getter/setter methods, and a `*classname*.java` file, which extends the data file, but is more or less empty. While the data file is overwritten each time you newly generate code from the ontology, the other file can be edited (e.g., to add custom methods), because changes will be preserved.

Fixed
: option only creates one file for each ontology class. This file should not be edited, because changes are lost, when regenerating code.

7.4. Class Options

The Protégé™ classes panel is extended with extra options concerning the code generation. These extra options are shown below the template slots list (see Figure 7.3, “Protégé™ classes panel”, bottom right). As default, the Protégé™ name of the class is used for the .java file. This can be overridden by specifying an additional Java class name. The interface flag can be set, when not a class but an interface should be generated. In general, this only makes sense for abstract ontology classes.



Relationship: Superclass

Name: Pet

Documentation: Abstract base class for pets.

Constraints:

Role: Concrete

Template Slots

Name	Type	Cardinality	Other Facets
age	Integer	single	
name	String	required single	

Class Name: AbstractPet

☐ Interface

Superclass:

Package:

Additional Interfaces: com.mypetstore.IPet

Generation Options: Editable

Figure 7.3. Protégé™ classes panel

The package field allows to specify the package of the Java class, overriding the default package specified in the code generation tab. The Generation Options offer the same options as the Files to Genrate, and influence how many files are generated for each ontology class. E.g., setting the option to Fixed allows to include an already existing class in the ontology. In addition, the field can be left blank to indicate that the ontology default should be used. The superclass field and the additional interfaces list, allow to specify fully qualified class or interface names to use as superclass or additionally implemented interfaces. If no superclass is given, the code generator creates a class hierarchy corresponding to the hierarchy in the ontology. For interfaces, only `implements` *`interface`*`, ...` is added to the class, the generator does not (yet) magically fill in any missing method implementations.

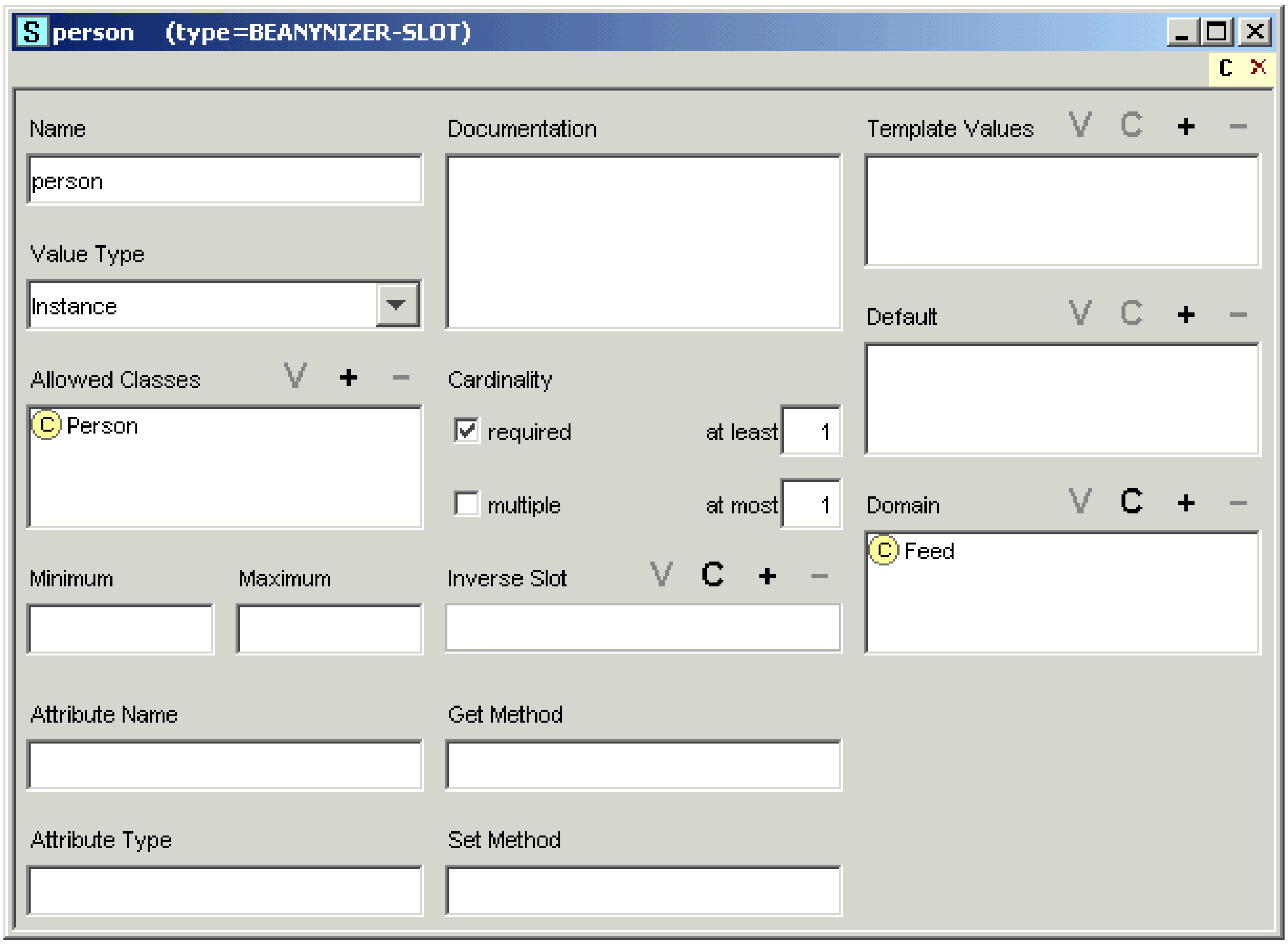
7.5. Slot Options

The code generator generates fields and getter / setter methods for each slot, thereby respecting settings such as name, type, default value, and cardinality (see Figure 7.4, "Protégé™ Slot Options"). For slots that allow multiple values, also add / remove methods are generated. Supported slot types and their default Java mappings are:

Table 7.1. Slot to Java type mappings

Slot Type	Java Type
Any	n/a
Boolean	`boolean`
Class	n/a

Slot Type	Java Type
Float	`double`
Instance	a java class
Integer	`int`
String	`java.lang.String`
Symbol	`java.lang.String`


S person (type=BEANYNIZER-SLOT)

Name: person

Documentation:

Template Values:

Value Type: Instance

Default:

Allowed Classes: Person

Cardinality: [✓ required] at least 1; multiple at most 1

Domain: Feed

Minimum:

Maximum:

Inverse Slot:

Attribute Name:

Get Method:

Attribute Type:

Set Method:

Figure 7.4. Protégé™ Slot Options

The name to use for the generated field can be specified using the attribute name option. The Beanynizer has its own idea of Java coding conventions and tries to create a suitable Java name from the slot name, when no specific attribute name is given. The attribute type allows to change the generated Java type, by specifying a fully qualified class name, or one of the basic types (e.g. long). The names of getter and setter methods are derived from the attribute name (which may also be derived from the slot name). Use the get method and set method options to change the names of the methods to generate. This is especially helpful, when including already existing classes in the ontology. The pure Java ontology supports two other settings for slots: transient and external. For transient slots, the field is generated with the `transient` keyword. External slots are ignored during code generation (this is useful, if the ontology class extends an existing Java class, which already provides the get/set methods for a given slot).

7.6. Converting an Existing Ontology

It is possible (while maybe a bit awkward) to convert other ontologies to be used with the Beanynizer. To convert an existing ontology perform the following steps (note, this process has only been tested with Protégé 2.1 and might not work with 3.0):

- Load the old ontology and save it under a new name (to keep the original file untouched).
- Use the Project → Merge included projects option to remove references to other external ontologies.
- Include the Beanynizer default ontology as described above. If your original ontology was also designed for use with a FIPA-compliant agent platform, ignore any errors, e.g., complaining about duplicate definition of classes like `AgentAction`, etc.
- If your original ontology did contain FIPA classes with different names (e.g., `AID` for agent identifiers) change all references to these classes (if any) to now refer to the Beanynizer classes (e.g., `agent-identifier`). Afterwards remove the original FIPA classes.
- When your original ontology was not designed to be used with FIPA, you might have to rearrange the class hierarchy, such that all your classes are derived from the appropriate classes (such as AgentAction).
- Now the awkward part: Make sure that all your classes and slots are instances of the Beanynizer metaclasses. The required metaclass is called `BEANYNIZER-CLASS` and is a subclass of the `:STANDARD-CLASS`. You change the metaclass by selecting each single class and using the Change metaclass... option from the popup menu, but a much faster way is to change to the instance tab, and use drag and drop. The same procedure should be done for your slots, which should be instances of the `BEANYNIZER-SLOT`. Finally, you should make the Beanynizer class and the Beanynizer slot the default metaclasses by using the Set as default class/slot meta class option from the popup menu.
- You're done! You can now start to generate code, or to adjust code generation options as described above.

7.7. Final Notes

Protégé™ is a complex tool in itself, therefore a basic understanding of it is essential before you can effectively use the Beanynizer plugin. The Beanynizer is an early staged recent development based on our specific requirements, and does not try to be a general purpose code generation environment. If you encounter problems or miss some features please drop us a note, such that we can improve the Beanynizer for upcoming releases.

Usages of the Beanynizer can be found in the cleanerworld, marsworld and hunterprey examples (look for an "ontology "package). Also, some Jadex tools use a Beanynizer generated ontology for communication (e.g., logger, introspector, and tracer). Their Protégé™ ontology files can be found in the `jadex/onto` directory.

The Beanynizer was designed for flexible code generation. The code for the Java classes is based on templates processed with the Velocity template engine. The templates for the Java and Jade generation modes can be found in the `src/jadex/tools/beanynizer/genjava` and `src/jadex/tools/beanynizer/genjade` directories. If you want to change the way Beanynizer generates code, you may try to alter these templates to suit your needs. See http://jakarta.apache.org/velocity/ for more information about the Velocity template language.

Chapter 8. Jadexdoc Tool

The Jadexdoc Tool is a documentation tool similar to the Javadoc tool. It provides the ability to generate HTML pages for ADF (Agent Description File) documentation from Jadex source files. The Jadexdoc tool parses the declarations and documentation comments in a set of agent description files and produces a corresponding set of HTML pages describing the agents, capabilities, beliefs, goals, plans, events and expressions. You can use it to generate the ADF documentation or the implementation documentation for a set of agents and capabilities. You can run the Jadexdoc tool on entire packages, individual source files, or both. When documenting entire packages, you can either traverse recursively down from a top-level directory, or pass in an explicit list of package names. When documenting individual source files, you pass in a list of source (`.agent.xml` or `.capability.xml`) filenames.

8.1. Usage

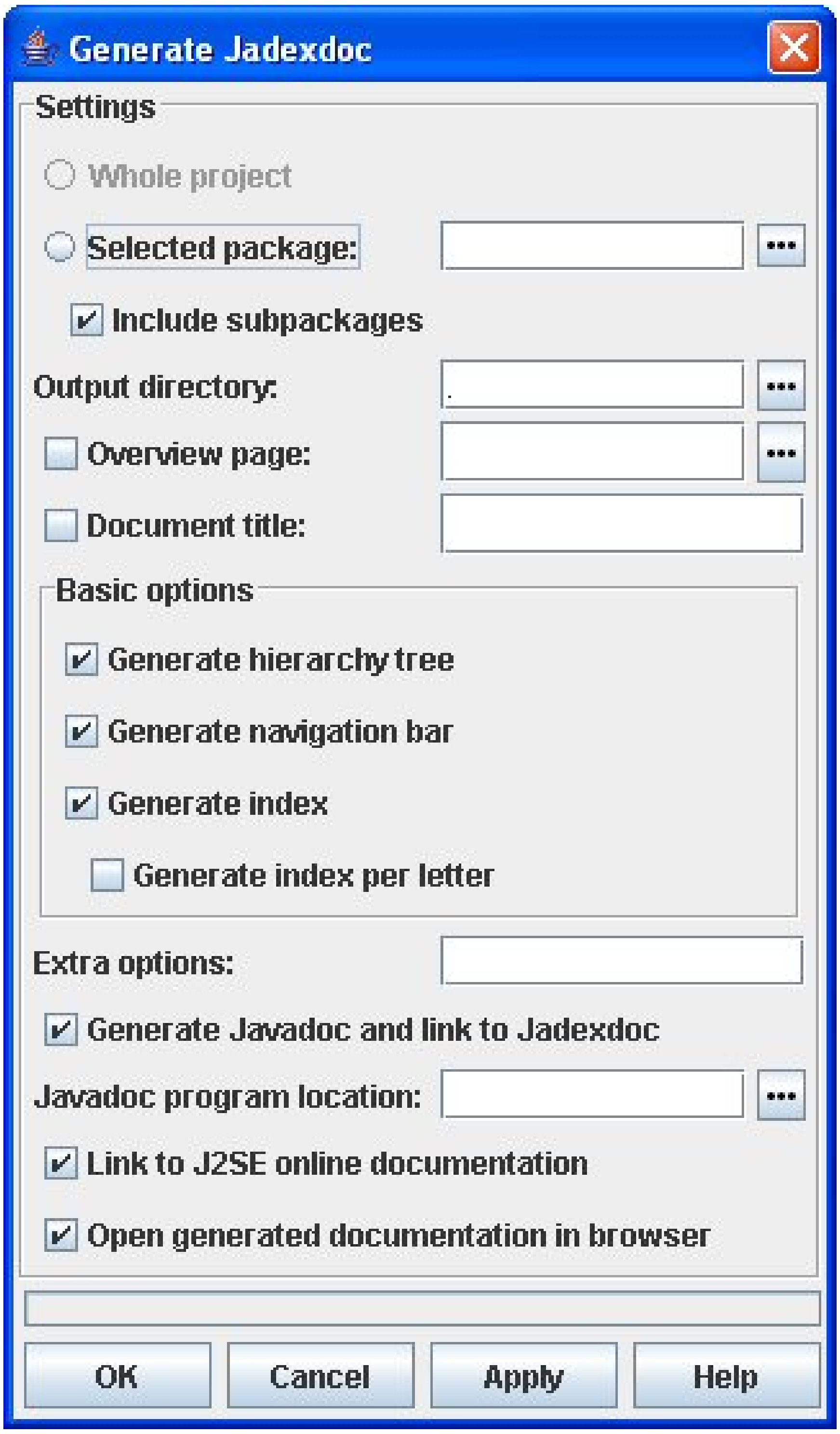
You can start the Jadexdoc tool from a console via:
java jadex.tools.jadexdoc.Main [options] [packagenames] [sourcefilenames] [-subpackage pck1:..]

Description. Arguments can be in any order. See processing of Source Files for details on how the Jadexdoc tool determines which source files to process.

- **options.** Command-line options, as specified in this document. The section options below contains examples of Jadexdoc options.
- **packagenames.** A series of names of packages, separated by spaces, such as `jadex.examples jadex.planlib`. You must separately specify each package you want to document. Wildcards are not allowed; use `-subpackages` for recursion. For further details see the section Section 8.5, “Options”.
- **sourcefilenames.** A series of source file names, separated by spaces, each of which can begin with a path and contain a wildcard such as asterisk (*). The Jadexdoc tool will process every file whose name ends with ".agent.xml" or ".capability.xml".
- **-subpackages** ***`pck1:pck2:...`*** Generates documentation from source files in the specified packages and recursively in their subpackages. An alternative to supplying packagenames or sourcefilenames.

Generation Dialog. It is also possible to start Jadexdoc via user interface. The command to start the user interface generation dialog (see Figure 8.1, “Generate Jadexdoc Dialog”) is:

java jadex.tools.jadexdoc.GenerateDialog
Another possibility to start the dialog is diectly from the Jadex Control Center. You can select an agent (or capability) or a package in the model explorer of the starter perspective. Clicking in the toolbar the "Generate Jadexdoc" will open the dialog in which the generation settings can be specified. After successful generation a browser will be opened and the generated documentation is shown.


Generate Jadexdoc

Settings

Whole project

Selected package:

[✓] Include subpackages

Output directory: .

[ ] Overview page:

[ ] Document title:

Basic options

[✓] Generate hierarchy tree

[✓] Generate navigation bar

[✓] Generate index

[ ] Generate index per letter

Extra options:

[✓] Generate Javadoc and link to Jadexdoc

Javadoc program location:

[✓] Link to J2SE online documentation

[✓] Open generated documentation in browser

Figure 8.1. Generate Jadexdoc Dialog

The dialog offers several settings that can be adjusted before the generation is started via the Ok or Apply buttons. The documentation may take a while so the progress of the generation process is shown in the progress bar above the buttons.

- **Whole project.** Documentation for the whole project and all contained packages will be created. This option is only available when the dialog is started from the Control Center.
- **Selected Package.** Documentation for the selected package or file is generated. If the tool is started from the Control Center the currently selected package will be shown. If the dialog is started via command-line

you can use the "..." button to select a file or directory to document.

- **Include subdirectories.** If turned on all subpackages of the selected package are automatically included in the generation process.
- **Output directory.** The standard output directory is the current directory. The "..." button can be used to select another appropriate target directory.
- **Overview page.** The overview page represents the top-level page for the whole generated documentation and contains information about the contained packages. The "..." button can be used to select a custom overview HTML page. The overview page will only be included when the corresponding check box is selected.
- **Document title.** Specifies the title to be placed near the top of the overview summary file. The title will be placed as a centered, level-one heading directly beneath the upper navigation bar. The title will only be included when the corresponding check box is selected.
- **Basic options.** The basic options can be used to turn on/off several generation features. The hierarchy tree is a page containing agents and capabilities displayed in usage relationships. The navigation bar offers possibilities to refer to related documentation pages. It can be turned off if you are interested only in the content and have no need for navigation, e.g. when converting the files to PostScript or PDF for print only. The index holds an alphabetical list of elements. It can be adjusted with an options in a way that only one letter per page is generated.
- **Extra options.** In the extra options text field an arbitrary number of additional Jadexdoc command line options can be specified.
- **Generate Javadoc and link to Jadexdoc.** If selected in a first run Javadoc will be invoked to produce the Javadoc documentation for the selected packages. In a second run Jadexdoc will additionally create the agent-related documentation and link it with the formerly produced class information.
- **Link to J2SE online documentation.** If selected the generated documentation will be connected to the online Javadoc of the Java Development Kit 1.5.
- **Link to J2SE online documentation.** If selected the generated documentation will be connected to the online Javadoc of the Java Development Kit 1.5.
- **Open generated documentation in browser.** If selected the generated documentation (`index.html`) will be opened in the default browser of the system.

8.2. Source Files

The Jadexdoc tool will generate output originating from four different types of "source" files: Agent and Capability source files, package comment files, overview comment files, and miscellaneous unprocessed files.

Processing of source files. The Jadexdoc tool processes files that end with ".agent.xml" and ".capability.xml" plus other files described below. If you run the Jadexdoc tool by explicitly passing in individual source filenames, you can determine exactly which ADF files are processed. However, that is not how most developers want to work, as it is simpler to pass in package names.

Package Comment Files. Each package can have its own documentation comment, contained in its own "source" file, that the Jadexdoc tool will merge into the package summary page that it generates. You typically include in this comment any documentation that applies to the entire package.

To create a package comment file, you must name it `package.html` and place it in the package directory in the source tree along with the agent description files. The Jadexdoc tool will automatically look for this filename in this location. Notice that the filename is identical for all packages.

The content of the package comment file is one big documentation comment, written in HTML, like all other comments. When writing the comment, you should make the first sentence a summary about the package, and not put a title or any other text between `<body>` and the first sentence.

When the Jadexdoc tool runs, it will automatically look for this file; if found, the Jadexdoc tool does the following:

- Copies all content between `<body>` and `</body>` tags for processing.
- Inserts the processed text at the bottom of the package summary page it generates, as shown in Package Summary.
- Copies the first sentence of the package comment to the top of the package summary page. It also adds the package name and this first sentence to the list of packages on the overview page.

Overview Comment File. Each application or set of packages that you are documenting can have its own overview documentation comment, kept in its own "source" file, that the Jadexdoc tool will merge into the overview page that it generates. You typically include in this comment any documentation that applies to the entire application or set of packages.

To create an overview comment file, you can name the file anything you want, typically `overview.html` and place it anywhere, typically at the top level of the source tree. The content of the overview comment file is one big documentation comment, written in HTML, like the package comment file described previously. When you run the Jadexdoc tool, you specify the overview comment file name with the `-overview` option. The file is then processed similar to that of a package comment file.

- Copies all content between `<body>` and `</body>` tags for processing.
- Inserts the processed text at the bottom of the overview page it generates.
- Copies the first sentence of the overview comment to the top of the overview summary page.

Miscellaneous Unprocessed Files. You can also include in your source any miscellaneous files that you want the Jadexdoc tool to copy to the destination directory. These typically includes graphic files, example agent description files, and self-standing HTML files whose content would overwhelm the documentation comment of a normal agent description file.

To include unprocessed files, put them in a directory called `doc-files` which can be a subdirectory of any package directory that contains source files. You can have one such subdirectory for each package. For example, if you want to include the image of a creature `Creature.png` in the `jadex.examples.hunterprey.creature.CleverPrey` agent documentation, you place that file in the `jadex/examples/hunterprey/creature/doc-files/` directory. Notice the `doc-files` directory should not be located at `jadex/examples/doc-files/` because `examples` does not directly contain any source files.

All links to these unprocessed files must be hard-coded, because the Jadexdoc tool does not look at the files, it simply copies the directory and all its contents to the destination. For example, the link in the `CleverPrey.agent.xml` doc comment might look like:

```
<!-- The image of the CleverPrey agent: <img src="doc-files/Creature.png"> -->
```

8.3. Generated Files

By default, Jadexdoc uses a standard doclet that generates HTML-formatted documentation. This doclet generates the following kinds of files (where each HTML "page" corresponds to a separate file). Note that Jadexdoc generates files with two types of names: those named after agents/capabilities, and those that are not (such as `package-summary.html`). Files in the latter group contain hyphens to prevent filename conflicts with those in the former group.

Basic Content Pages.

- One agent (`agentname.agent.html`) or capability (`capabilityname.capability.html`) page for each agent or capability is documented.
- One package page (`package-summary.html`) for each package it is documenting. The Jadexdoc tool will include any HTML text provided in a file named `package.html` in the package directory of the source tree.
- One overview page (`overview-summary.html`) for the entire set of packages. This is the front page of the generated document. The Jadexdoc tool will include any HTML text provided in a file specified with the `-overview` option. Note that this file is created only if you pass into Jadexdoc two or more package names.

Cross-Reference Pages.

- One *agent/capability* hierarchy page for the *entire set of packages* (`overview-tree.html`). To view this, click on Overview in the navigation bar, then click on Tree.
- One *agent/capability* hierarchy page for *each package* (`package-tree.html`). To view this, go to a particular package, agent or capability page; click Tree to display the hierarchy for that package.
- An index (`index-*.html`) of all agent, capabilities, beliefs, plans, goals, events and expressions names, alphabetically arranged.

Support Files.

- A *help page* (`help-doc.html`) that describes the navigation bar and the above pages. You can provide your own custom help file to override the default using `-helpfile`.
- One `index.html` file which creates the HTML frames for display. This is the file you load to display the front page with frames. This file itself contains no text content. Several frame files (`*-frame.html`) containing lists of packages, agents and capabilities, used when HTML frames are being displayed.
- A *style sheet file* (`stylesheet.css`) that controls a limited amount of color, font family, font size, font style and positioning on the generated pages.
- A `doc-files` directory that holds any image, example, source code or other files that you want copied to the destination directory.

8.4. Documentation Comments

Commenting the Source Code. You can include documentation comments in the agent description files, ahead of declarations for any agent, capability, plan, goal, event or expression, etc. You can also create comments for each package and another one for the overview, though their syntax is slightly different. The comments in the agent description files are regular xml-comments consisting of the characters between the characters `<!-- that begin the comment and the characters -->` that end it. The text in a comment can continue onto multiple lines.

```
<!--
    This is the typical format of a simple documentation
    comment that spans multiple lines.
-->

<!-- To save space you can also put a comment on one line. -->
```

Placement of comments. Documentation comments are recognized only when placed immediately before agent, capability, belief, goal, plan, event or expression declarations. Only one documentation comment per declaration statement is recognized by the Jadexdoc tool.

```
<!-- This is the comment for the agent 'MyAgent' -->
<agent name="MyAgent" package="jadex.examples.myagents">
    <beliefs>
        <!-- The comment for the belief 'MyBelief' -->
        <belief name="MyBelief" class="Object"/>
    </beliefs>
</agent>
```

Comments are written in HTML. The texts can be written in HTML, in that they should use HTML entities and can use HTML tags. You can use whichever version of HTML your browser supports. The bold HTML tag `<b>` is shown in the following example.

```
<!-- This is a <b>documentation</b> comment. -->
```

First sentence. The first sentence of each documentation comment should be a summary sentence, containing a concise but complete description of the declared member. This sentence ends at the first period that is followed by a blank, tab, or line terminator. The Jadexdoc tool copies this first sentence to the member summary at the top of the HTML page. For convenience, Jadexdoc strips any html tags from this sentence, when it is displayed in a summary table.

8.5. Options

The Jadexdoc tool uses a standard doclet to determine its output. The Jadexdoc tool provides a set of command-line options that can be used with any doclet. These options are described below under the sub-heading Section 8.5.1, “Jadexdoc Options”. The standard doclet provides an additional set of command-line options that are described below under the sub-heading Options Provided by the Standard Doclet. All option names are case-insensitive, though their arguments can be case-sensitive.

8.5.1. Jadexdoc Options

`-subpackages` *`pck1:pck2:...`*
: Generates documentation from source files in the specified packages and recursively in their subpackages. This option is useful when adding new subpackages to the source code, as they are automatically included.

Each package argument is any top-level subpackage (such as `jadex`) or fully qualified package (such as `jadex.examples`) that does not need to contain source files. Arguments are separated by colons (on all operating systems). Wildcards are not needed or allowed.

java jadex.tools.jadexdoc.Main -d docs -subpackages jadex.examples.hunterprey:jadex.examples.cleanerworld

You can use `-subpackages` in conjunction with `-exclude` to exclude specific packages.

`-exclude` *`pck1:pck2:...`*

Unconditionally excludes the specified packages and their subpackages from the list formed by `-subpackages`. It excludes those packages even if they would otherwise be included by some previous or later `-subpackages` option.

java jadex.tools.jadexdoc.Main -d docs -subpackages jadex -exclude jadex.planlib:jadex.examples.testcases

`-help`

Displays the online help, which lists these Jadexdoc and doclet command line options.

`-quiet`

Shuts off non-error and non-warning messages, leaving only the warnings and errors appear, making them easier to view. Also suppresses the version string.

8.5.2. Options Provided by the Standard Doclet

`-d` *`directory`*

Specifies the destination directory where Jadexdoc saves the generated HTML files. Omitting this option causes the files to be saved to the current directory. The value directory can be absolute, or relative to the current working directory. The destination directory is automatically created when Jadexdoc is run. For example, the following generates the documentation for the package `jadex.examples.testcases` and saves the results in the `user/doc` directory: **java jadex.tools.jadexdoc.Main -d user/doc jadex.examples.testcases**

`-overview` *`path/filename`*

Specifies that Jadexdoc should retrieve the text for the overview documentation from the "source" file specified by *`path/filename`* and place it on the Overview page (`overview-summary.html`). While you can use any name you want for filename and place it anywhere you want for path, a typical thing to do is to name it `overview.html` and place it in the source tree at the directory that contains the topmost package directories. Note that the overview page is created only if you pass into Jadexdoc two or more package names. The title on the overview page is set by `-doctitle`.

`-splitindex`

Splits the index file into multiple files, alphabetically, one file per letter, plus a file for any index entries that start with non-alphabetical characters.

`-windowtitle` *`title`*

Specifies the title to be placed in the HTML `<title>` tag. This appears in the window title and in any browser bookmarks (favorite places) that someone creates for this page. This title should not contain any HTML tags, as the browser will not properly interpret them. Any internal quotation marks within title may have to be escaped. If `-windowtitle` is omitted, the Jadexdoc tool uses the value of `-doctitle` for this option.

java jadex.tools.jadexdoc.Main -windowtitle "Jadex Examples" jadex.examples

`-doctitle` *`title`*

Specifies the title to be placed near the top of the overview summary file. The title will be placed as a centered, level-one heading directly beneath the upper navigation bar. The title may contain html tags and white space, though if it does, it must be enclosed in quotes. Any internal quotation marks within title may have to be escaped.

java jadex.tools.jadexdoc.Main -doctitle "<b>Jadex Agent Dokumentation<b>" jadex.examples.testcases

`-header` *`header`*

Specifies the header text to be placed at the top of each output file. The header will be placed to the right of the upper navigation bar. *`header`* may contain HTML tags and white space, though if it does, it must be enclosed in quotes. Any internal quotation marks within header may have to be escaped.

java jadex.tools.jadexdoc.Main -header "<b>Jadex Platform</b><br>0.93" jadex.examples.testcases

`-footer` *`footer`*

Specifies the footer text to be placed at the bottom of each output file. The footer will be placed to the right of the lower navigation bar. *`footer`* may contain html tags and white space, though if it does, it must be enclosed in quotes. Any internal quotation marks within footer may have to be escaped.

`-bottom` *`text`*

Specifies the text to be placed at the bottom of each output file. The text will be placed at the bottom of the page, below the lower navigation bar. The text may contain HTML tags and white space, though if it does, it must be enclosed in quotes. Any internal quotation marks within text may have to be escaped.

`-link` *`extdocURL`*

The `-link` option enables you to link to java classes referenced to by your members in the agent description file. For these links to go to valid pages, you must know where those HTML pages are located, and specify that location with extdocURL. This allows, for instance, a jadex doc file to link to java.* documentation on http://java.sun.com. When Jadexdoc is run without the `-link` option, when it encounters a java class, it prints the fully qualified name with no link. However, when the `-link` option is used, the Jadexdoc tool searches the `package-list` file at the specified *`extdocURL`* location for that package name. If it finds the package name, it creates a link to the external javadoc location.

`extdocURL` is the absolute or relative URL of the directory containing the external javadoc-generated documentation you want to link to. Examples are shown below. The package-list file must be found in this directory (otherwise, use `-linkoffline`). The Jadexdoc tool reads the package names from the package-list file and then links to those packages at that URL. When the Jadexdoc tool is run, the *`extdocURL`* value is copied literally into the <A HREF> links that are created. Therefore, *`extdocURL`* must be the URL to the directory, not to a file. You can use an absolute link for extdocURL to enable your docs to link to a document on any website, or can use a relative link to link only to a relative location. If relative, the value you pass in should be the relative path from the destination directory (specified with `-d`) to the directory containing the packages being linked to. In all cases, and on all operating systems, you should use a forward slash as the separator, whether the URL is absolute or relative, and "http:" or "file:" based (as specified in theURL Memo).

Absolute *http:* based link:

```
-link http://<host>/<directory>/<directory>/.../<name>
```

Absolute *file:* based link:

```
-link file://<host>/<directory>/<directory>/.../<name>
```

Relative link:

```
-link <directory>/<directory>/.../<name>
```

You can specify multiple `-link` options in a given Jadexdoc run to link to multiple documents.

Choosing between `-linkoffline` and `-link`. Use `-link`: when using a relative path to the external API document, or when using an absolute URL to the external API document, if your shell allows a program to open a connection to that URL for reading.

Use `-linkoffline`: when using an absolute URL to the external API document, if your shell does not allow a program to open a connection to that URL for reading. This can occur if you are behind a firewall and the document you want to link to is on the other side.

Example 8.1. Example using absolute links to the external docs

Let's say you want to link to the Java 2 Platform packages at http://java.sun.com/j2se/1.5.0/docs/api/. The following command generates documentation for the package `jadex.examples` with links to the Java 2 Platform packages.

java jadex.tools.jadexdoc.Main -link http://java.sun.com/j2se/1.5.0/docs/api/ -subpackages jadex.examples

Example 8.2. Example using relative links to the external docs

Let's say you have user defined java packages whose docs are generated with the Javadoc tool. Then you use the Jadexdoc tool to document the corresponding agent description files and those docs are separated by a relative path. In this example, the API (Application Programming Interface) packages reside in `docs/api/jadex/examples` and the ADF (agent description files) packages in `docs/adf/jadex/examples`. Assuming the API package documentation is already generated, and that docs is the current directory, you would document the ADF package with links to the API documentation by running:

java jadex.tools.jadexdoc.Main -d ../adf -link ../api -subpackages jadex.examples

The `-link` argument is relative to the destination directory. In order to avoid broken links, all of the documentation for the external references must exist at the specified URLs. The Jadexdoc tool will not check that these pages exist only that the package-list exists.

Multiple Links. You can supply multiple `-link` options to link to any number of external generated documents. Specify a different link option for each external document to link to:

java jadex.tools.jadexdoc.Main -link *`extdocURL1`* -link *`extdocURL2`* ... -subpackages jadex.examples
where *`extdocURL1`*, *`extdocURL2`*, ... point respectively to the roots of external documents, each of which contains a file named `package-list`.

`-linkoffline` *`extdocURL packagelistLoc`*
This option is a variation of `-link`; they both create links to javadoc-generated documentation for external referenced classes. Use the `-linkoffline` option when linking to a document on the web when the Javadoc tool itself is "offline" that is, it cannot access the document through a web connection.

The `-linkoffline` option takes two arguments the first for the string to be embedded in the <a href> links, the second telling it where to find package-list:

- *`extdocURL`* is the absolute or relative URL of the directory containing the external javadoc-generated documentation you want to link to. If relative, the value should be the relative path from the destination directory (specified with `-d`) to the root of the packages being linked to. For more details, see *`extdocURL`* in the `-link` option.

- *`packagelistLoc`* is the path or URL to the directory containing the package-list file for the external documentation. This can be a URL (http: or file:) or file path, and can be absolute or relative. If relative, make it relative to the current directory from where javadoc was run. Do not include the `package-list` filename.

You can specify multiple `-linkoffline` options in a given Jadexdoc run.

`-group` *`groupheading packagepattern:packagepattern:...`*

Separates packages on the overview page into whatever groups you specify, one group per table. You specify each group with a different `-group` option. The groups appear on the page in the order specified on the command line; packages are alphabetized within a group. For a given `-group` option, the packages matching the list of packagepattern expressions appear in a table with the heading *`groupheading`*.

`groupheading` can be any text, and can include white space. This text is placed in the table heading for the group.

`packagepattern` can be any package name, or can be the start of any package name followed by an asterisk (*). The asterisk is a wildcard meaning "match any characters". This is the only wildcard allowed. Multiple patterns can be included in a group by separating them with colons (:).

If using an asterisk in a pattern or pattern list, the pattern list must be inside quotes, such as "jadex.examples*"
If you do not supply any `-group` option, all packages are placed in one group with the heading *Packages*. If the groups do not include all documented packages, any leftover packages appear in a separate group with the heading *Other Packages*.

java jadex.tools.jadexdoc.Main -group "Core Packages" "jadex.planlib" -group "Hunterprey Packages" "jadex.examples.hunterprey*" -subpackages jadex.examples

`-notree`

Omits the agent/capability hierarchy pages from the generated docs. These are the pages you reach using the "Tree" button in the navigation bar. The hierarchy is produced by default.

`-noindex`

Omits the index from the generated docs. The index is produced by default.

`-nohelp`

Omits the "Help" link in the navigation bars at the top and bottom of each page of output.

`-nonavbar`

Prevents the generation of the navigation bar, header and footer, otherwise found at the top and bottom of the generated pages. Has no affect on the "bottom" option. The `-nonavbar` option is useful when you are interested only in the content and have no need for navigation.

`-helpfile` *`path/filename`*

Specifies the path of an alternate help file *`path/filename`* that the "Help" link in the top and bottom navigation bars link to. Without this option, the Jadexdoc tool automatically creates a help file `help-doc.html`.

This option enables you to override this default. The filename can be any name. The Jadexdoc tool will adjust the links in the navigation bar accordingly.

java jadex.tools.jadexdoc.Main -helpfile C:\user\myhelp.html -subpackages jadex.examples

`-stylesheetfile` *`path/filename`*

Specifies the path of an alternate HTML stylesheet file. Without this option, the Jadexdoc tool automatically creates a stylesheet file `stylesheet.css`. This option enables you to override this default. The filename can be any name.

java jadex.tools.jadexdoc.Main -stylesheetfile C:\user\mystylesheet.css -subpackages jadex.examples

`-docfilessubdirs`

Enables deep copying of "doc-files" directories. In other words, subdirectories and all contents are recursively copied to the destination. For example, the directory `doc-files/example/images` and all its contents would now be copied. There is also an option to exclude subdirectories.

`-excludedocfilessubdir` *`name1:name2...`*

Excludes any `doc-files` subdirectories with the given names.

`-noqualifier` *`all | packagename1:packagename2:...`*

Omits qualifying package name from ahead of agent/capability *`and`* class/interface names in output. The argument to `-noqualifier` is either `all` (all package qualifiers are omitted) or a colon-separated list of packages, with wildcards, to be removed as qualifiers. The package name is removed from places where agent/capability or class/interface names appear.

`-nocomment`

Suppress the entire comment body, including the main description, generating only declarations.

Appendix A. Jadex Remote Monitoring Agent

JADE is distributed with a graphical control center called Remote Monitoring Agent (RMA). An extended version of the RMA is included with Jadex. The class of the agent is`jadex.adapter.jade.tools.rma`. The Jadex RMA is mainly a clone of the JADE RMA (cf. Figure A.1, "Jadex RMA Main Window"). The extended RMA provides shortcut icons for starting the new tool agents:

- Jadex Control Center
- Logger

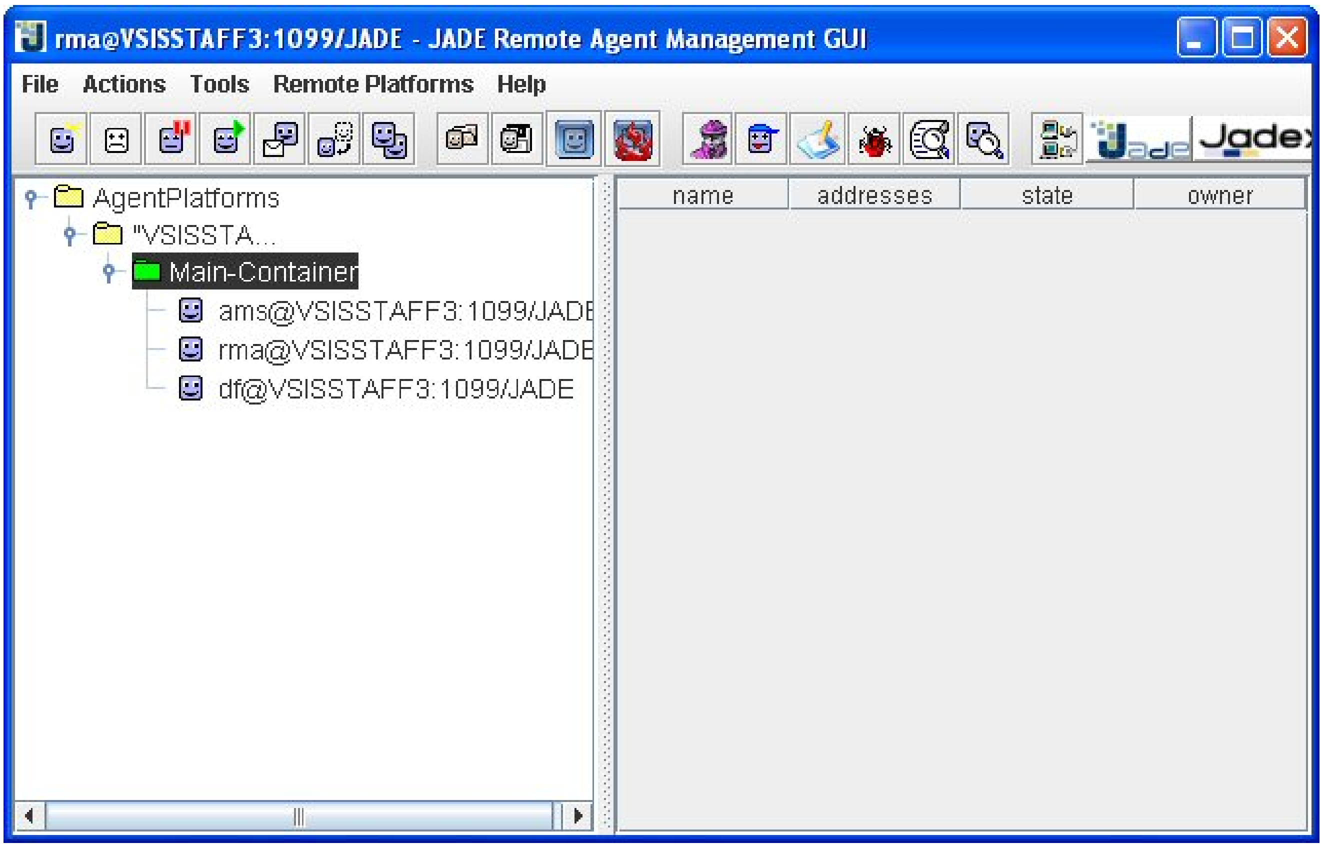
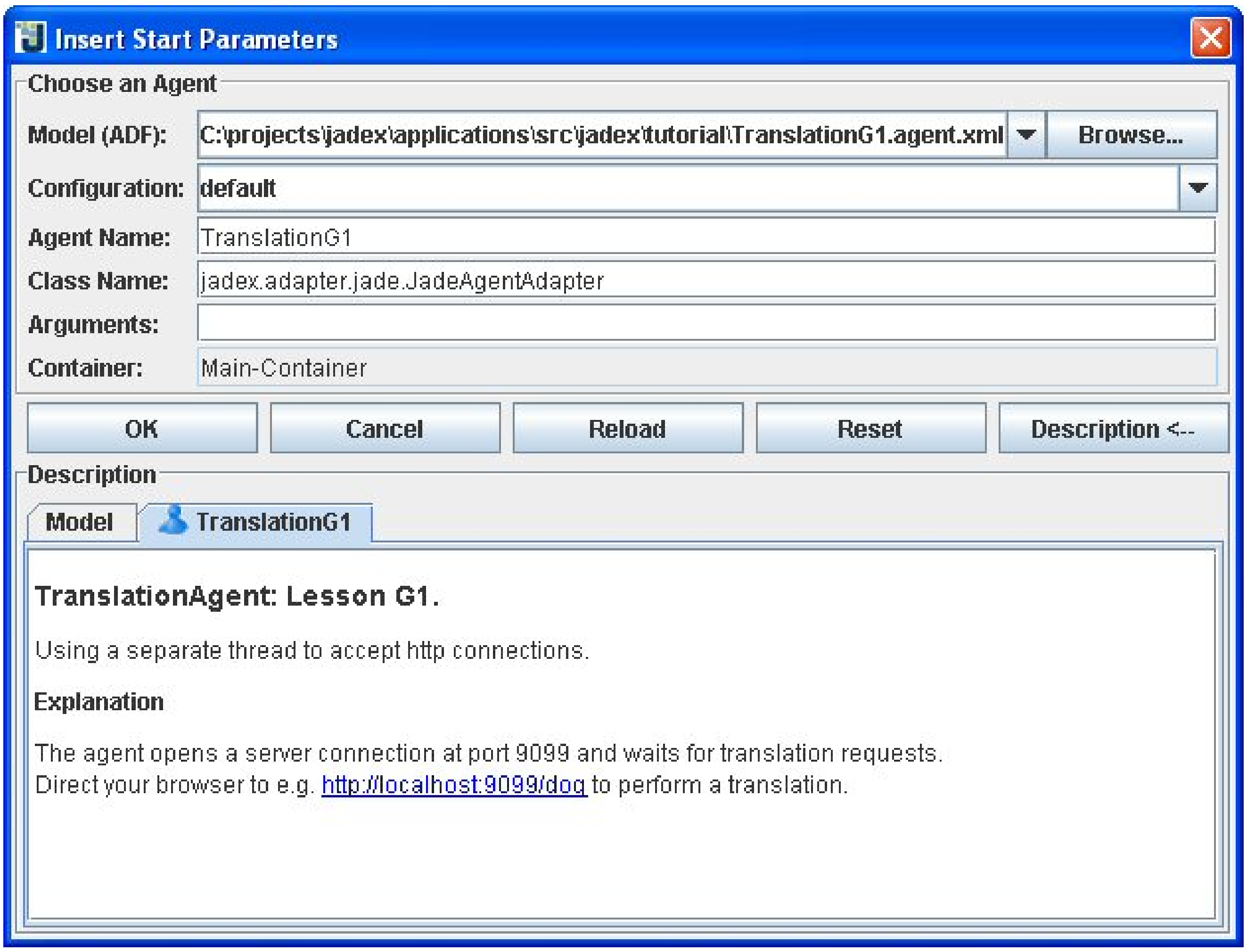


Figure A.1. Jadex RMA Main Window

A.1. Start New Agent Dialog

The main difference to the JADE RMA is the new dialog for starting agents (seeFigure A.2, "Agent Start Dialog"). With the Jadex RMA, an agent definition file (ADF) can be selected for execution. This ADF is supplied either by entering the file name in the Model (ADF) textfield, or by selecting the file from a file requester (Browse... button). The configuration choice box contains all available start configuration of the loaded agent. After loading the default configuration is automatically selected. The other start parameters (Agent Name, Class Name, Arguments, Container) are the same as in the JADE RMA.



Model (ADF): C:\projects\jadex\applications\src\jadex\tutorial\TranslationG1.agent.xml

Configuration: default

Agent Name: TranslationG1

Class Name: jadex.adapter.jade.JadeAgentAdapter

Arguments:

Container: Main-Container

TranslationAgent: Lesson G1.

Using a separate thread to accept http connections.

Explanation

The agent opens a server connection at port 9099 and waits for translation requests.
Direct your browser to e.g. http://localhost:9099/dog to perform a translation.

Figure A.2. Agent Start Dialog

When loading an agent model from an ADF, the agent name and agent class values are filled in automatically. In addition to the Ok and Cancel buttons for starting the agent or aborting the dialog, the Jadex RMA has three more buttons: Reload will read again the currently specified agent model (ADF), e.g. after you have edited the file to correct errors. The Jadex RMA keeps a list of recently loaded agents (stored in a file called rma.properties). This list can be cleared with the Reset button. The Description > button allows viewing a description (see below) of the loaded agent model, before launching the agent.

When an error occurs while loading an agent model, the error message is also displayed below the start parameters in the Description panel. In addition, the panel keeps available the tabs of the last loaded agent models.

A.2. Loading Agents from Jar Files

Since release 0.92 the Jadex RMA is capable of directly loading agent models from jar files. Use the file requester opened by the Browse... button to select a jar file. The RMA will inspect the jar file and allow to choose one of the ADFs that are contained in the jar file (see Figure A.3, “ JAR Selection Dialog ”). The RMA will also add the jar file to the class loader used by Jadex, so any resources from this jar (classes, images, etc.) can be used by the loaded agent, even if the jar file is not contained in the class path. If you want to clean the classpath used by the rma, you can either manually edit the rma properties or just hit the start dialog's reset button.

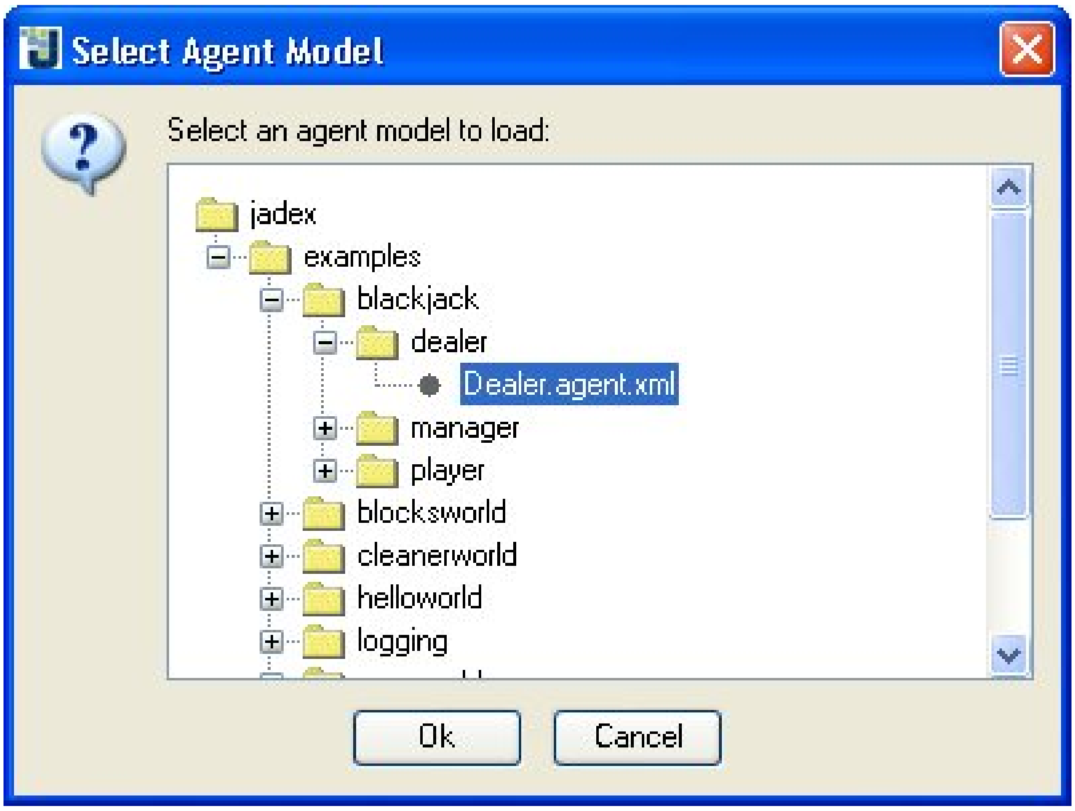


Figure A.3. JAR Selection Dialog

Appendix B. Logger

The Logger Agent, available from the RMA's tools menu, allows viewing log messages generated by other agents. There are three main aspects described below. First there is a brief description of the Logger Agent's functionality. The next section specifies how to customize JADE and Jadex agents to generate observable log messages and finally it is shown how to use the Logger Agent to view log messages from observed agents.

B.1. Logging Overview

The implementation of the Logger Agent is based on The Java™ Logging API (`java.util.logging`). Agents make logging calls on `Logger` objects. These objects allocate `LogRecord` objects which are passed to notifier agents via special `Handler` objects on demand. The notifier agents themselves are responsible for sending an appropriate ACL message to the Logger Agent for displaying the log message.

Both loggers and handlers may use logging levels to decide if they are interested in a particular log record. The level gives a rough guide to the importance and urgency of a log message. The Level class defines seven standard log levels, ranging from FINEST (the lowest priority) to SEVERE (the highest priority).

Loggers are organized in a hierarchical namespace and child loggers may inherit some logging properties from their parents in the namespace. Loggers are normally named entities, using dot-separated names such as java package names, but for the agent's purpose they are just named like the agents themselves. Therefore the only parent from which the loggers may inherit logging properties is the root logger (named ""). But Loggers may also be configured to ignore handlers higher up the tree.

There is a default logging configuration that ships with the Java Runtime Environment, and it can be overridden by software vendors, system admins, and end users. The default configuration establishes a single handler on the root logger for sending output to the console with log levels INFO and higher.

The Logger class provides a large set of methods for generating log messages. For convenience, there are methods for each logging level, named after the logging level. There are also methods that take an explicit source class name and source method name to quickly locate the source of any given logging message. Methods without it make a "best effort" to determine which class and method has called it and will add this information into the log record. For further details on Java™ Logging APIs see the corresponding documentation.

B.2. Generate Log Messages

The generation of log messages with Jadex agents is quite easy. First you need a reference to the agent's `Logger` object that can be acquired from an agent's plan helper method. On this object you can call logging methods with an associated log level.

```
// get the agent's logger
Logger logger= this.getLogger();
// Log simple INFO message
logger.log(Level.INFO, "doing stuff");
```

For customizing purpose you can define logging properties in the agent description file (ADF). You can set the log level for the agent's logger object and specify if it should inherit the logging properties of the root logger.

```
<properties>
<!-- Request all Details -->
<property name = "logging.level">Level.ALL</property>
<!-- use Handlers of parent (root) logger -->
<property name = "logging.useParentHandlers">true</property>
</properties>
```

For JADE agents there are neither helper methods to get the agent's logger object nor customizing possibilities using property files. But anyways, it is quite simple to use the logging functionality from JADE agents. The following example explains how to use logging methods in JADE.

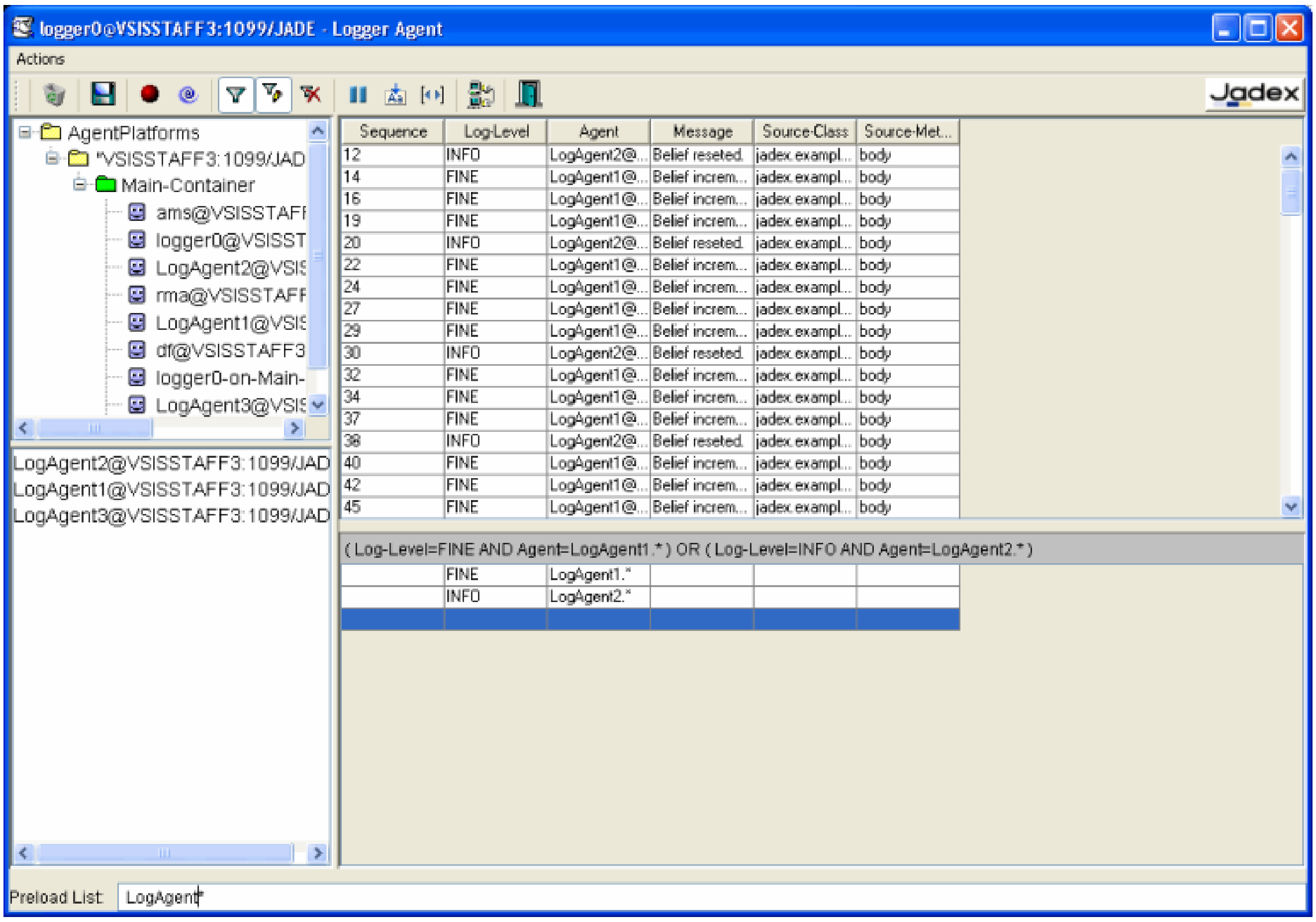
```
// get logger with agent's name
String name = getName();
logger = Logger.getLogger(name);
// Request all Details
logger.setLevel(Level.ALL);
// use Handlers of parent (root) logger
logger.setUseParentHandlers(true);
// Log simple INFO message
logger.log(Level.INFO, "doing stuff");
```

For examples of agents with logging functionality see code provided in the `src/examples/logging directory`.

B.3. View Log Messages

The usage of the Logger Agent is similar to the JADE Sniffer Agent. The Logger Agent can be started either from the tools menu of the RMA or from the command line as follows:
`java jade.Boot` {logger:jadex.tools.logger.Logger}

The main window contains four distinct areas, separated by adjustable split panels (cf. Figure B.1, “Main frame of Introspector”) . On the left hand side there is the Selection Agents Window showing the available agents on the platform. This is the place where you can choose the agents you want to observe. Observed agent are presented in the Agent List Window below.



Sequence	Log-Level	Agent	Message	Source-Class	Source-Met...
12	INFO	LogAgent2@...	Belief reseted.	jadex.exampl...	body
14	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
16	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
19	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
20	INFO	LogAgent2@...	Belief reseted.	jadex.exampl...	body
22	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
24	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
27	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
29	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
30	INFO	LogAgent2@...	Belief reseted.	jadex.exampl...	body
32	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
34	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
37	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
38	INFO	LogAgent2@...	Belief reseted.	jadex.exampl...	body
40	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
42	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body
45	FINE	LogAgent1@...	Belief increm...	jadex.exampl...	body

(Log-Level=FINE AND Agent=LogAgent1.*) OR (Log-Level=INFO AND Agent=LogAgent2.*)

	FINE	LogAgent1.*			
	INFO	LogAgent2.*			

Figure B.1. Main frame of Introspector

When the user decides to log an agent every log message is tracked and displayed in the Log Table on the upper right hand side. The table's columns showing the properties of each log message created by observed agents. You can adjust column widths by resizing the table's header cells or double-clicking on the right end of them. A right click on the table's header provides a pop-up menu that allows making individual columns visible or invisible. Sorting is activated by clicking on a column header cell. The sort order is indicated by a little arrow. Following data of the log messages are presented in the Log Table. Some of them are suppressed by default:

Sequence
: The sequence property will be initialized with a unique value. Note that these sequence values are allocated in increasing order within a virtual machine (VM), so observing different agents on different containers or platforms may produce duplicated values in the table.

Date/Time
: The date/time property will be initialized to the current time when the log message is created.

Log-Level
: The given log level for the log message.

Agent
: The source agent's name.

Message
: The message text.

Source-Class
The class name that called into the logging framework.

Source-Method
The method name that called into the logging framework.

Thread-ID
The thread ID property will be initialized with a unique ID for the current thread.

The main intention of building the Logger Agent was the ability to provide filter techniques for customizing the view of log messages. Underneath the Log Table you can find the Filter Table which is visually synchronized in terms of column widths and visibility. For each column you can define one or more filter expressions. Any expression in a row is combined with the boolean AND operator and the rows are combined with the boolean OR operator. The whole boolean like expression is displayed in the text field above the table. A single filter expressions is a regular-expression which is matched against the data in the corresponding column. Regular-expressions are case-sensitive and there are several constructs to build patterns. A summary of regular-expression constructs can be found in the java.util.regex.Pattern class. For a short summary consult Table B.1, “Summary of Regular Expressions”

Table B.1. Summary of Regular Expressions

Construct	Matches
`x`	The character `x`
`[abc]`	`a`, `b`, or `c` (simple class)
`[^abc]`	Any character except `a`, `b`, or `c` (negation)
`[a-zA-Z]`	`a` through `z` or `A` through `Z`, inclusive (range)
`.`	Any character (may or may not match line terminators)
`\d`	A digit: `[0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\w`	A word character: `[a-zA-Z_0-9]`
`X ?`	`X`, once or not at all
`X *`	`X`, zero or more times
`X +`	`X`, one or more times
`XY`	`X` followed by `Y`
`X | Y`	Either `X` or `Y`
Examples of regular-expressions	
`LogAgent.*`	Any agent with the name starting with `LogAgent.`
`LogAgent\d+.*`	Any agent with the name starting with LogAgent followed by one or more digits and maybe some more arbitrary characters.

At the very bottom of the main window a Preload List input field is provided. It can be filled with a list of preload descriptions separated by a semicolon. Each description consists of an agent name match string. If there is no @ in the agent name, it assumes the current HAP for it. There are two characters with special significance. The '?' is a wild-card for any character. And the '*' for any substring (one ore more characters).

```
LogAgent1;LogAgent2
LogAgent*
*
```

The same functionality is provided via command line arguments for the Logger Agent.
`java jade.Boot` {logger:jadex.tools.logger.Logger(LogAgent*)}

A property file called `logger.properties` may also be used to control the logger properties. It is looked for in the current directory, and if not found, the agent looks in the parent directory and continues this until the file is either found or there isn't a parent directory. Note that any changes to the Preload List in the GUI won't change the preload property in the file.

```
Preload=LogAgent1;LogAgent2
```

The main window has a toolbar with further options to customize the view of log messages. Although the most buttons are rather self explaining here is a short description of every item.

Table B.2. Main Window Toolbar

🗑	Clears the Log Table erasing all the data stored in memory.
💾	Writes a semicolon-separated text file with all the log messages appearing in the Log Table.
●	Starts logging the selected agent(s) from the Selection Agents Window.
@	Stops logging the selected agent(s) from the Selection Agents Window.
▽	Enables or disables the filter provided by the Filter Table.
▽✎	Shows or hides the Filter Table to maximize the space for the Log Table.
▽✗	Clears all the data appearing in the Filter Table.
⏸	Pauses or resumes the displaying of new log messages. If displaying is paused new log messages will be tracked and stored in memory. The new data will be added to the Log Table at once if displaying is resumed.

	If autoscroll is enabled every new log message added to the Log Table will be selected and the table will scroll to the region making the new data visible.
	Automatically adjusts all column widths to fit the data in the table.
	This action allows getting the description of a remote Agent Platform via the remote AMS. After that right click on the added platform in the Agent Selection Window and Refresh Agent List to view the agents on the target platform. This makes it possible to log agents even from a remote Platform.
	Closes the GUI and kills the Logger Agent.

Bibliography

[Bratman 1987] M. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press. Cambridge, MA, USA. 1987.

[Braubach et al. 2004] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. *Goal Representation for BDI Agent Systems*. R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Proceedings of the Second Workshop on Programming Multiagent Systems: Languages, frameworks, techniques, and tools (ProMAS04)*. Springer. Berlin, New York. 2004. pp.9-20.

[Braubach et al. 2005a] L. Braubach, A. Pokahr, and W. Lamersdorf. . R. Unland, M. Klusch, and M. Calisti. *Software Agent-Based Applications, Platforms and Development Kits*. Birkhäuser. 2005. pp.143-168.

[Braubach et al. 2005b] L. Braubach, A. Pokahr, and W. Lamersdorf. *Extending the Capability Concept for Flexible BDI Agent Modularization*. R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Proceedings of the Third International Workshop on Programming Multi-Agent Systems (ProMAS'05)*. . 2005. pp.99-114.

[Busetta et al. 2000] P. Busetta, N. Howden, R. Rönnquist, and A. Hodgson. *Structuring BDI Agents in Functional Clusters*. N. Jennings and Y. Lespérance. *Intelligent Agents VI, Proceedings of the 6th International Workshop, Agent Theories, Architectures, and Languages (ATAL) '99*. Springer. Berlin, New York. 2000. pp.277-289.

[Hindriks et al. 1999] K. Hindriks, F. de Boer, W. van der Hoek, and J.-J. Meyer. *Agent Programming in 3APL*. N. Jennings, K. Sycara, and M. Georgeff. *Autonomous Agents and Multi-Agent Systems*. Kluwer Academic publishers. 1999. pp. 357-401.

[Huber 1999] M. Huber. *JAM: A BDI-Theoretic Mobile Agent Architecture*. O. Etzioni, J. Müller, and J. Bradshaw. *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS-99)*. ACM Press. New York. 1999. pp. 236-243.

[Jadex Tutorial] L. Braubach, A. Pokahr, and A. Walczak. *Jadex Tutorial*. 2005.

[Jadex Tool Guide] A. Pokahr, L. Braubach, R. Leppin, and A. Walczak. *Jadex Tool Guide*. 2005.

[Jadex User Guide] A. Pokahr, L. Braubach, and A. Walczak. *Jadex User Guide*. 2005.

[Lehman et al. 1996] J. F. Lehman, J. E. Laird, and P. S. Rosenbloom. *A gentle introduction to Soar, an architecture for human cognition*. *Invitation to Cognitive Science Vol. 4*. MIT press. 1996.

[McCarthy et al. 1979] J. McCarthy. *Ascribing mental qualities to machine*. M. Ringle. *Philosophical Perspectives in Artificial Intelligence*. Humanities Press. Atlantic Highlands, NJ. 1979. pp. 161-195.

[Pokahr et al. 2005a] A. Pokahr, L. Braubach, and W. Lamersdorf. *A Goal Deliberation Strategy for BDI Agent Systems*. T. Eymann, F. Klügl, W. Lamersdorf, M. Klusch, and M. Huhns. *In Proceedings of the third German conference on Multi-Agent System TEchnologieS (MATES-2005)*. Springer-Verlag. Berlin Heidelberg New York. 2005.

[Pokahr et al. 2005b] A. Pokahr, L. Braubach, and W. Lamersdorf. *A Flexible BDI Architecture Supporting Extensibility*. A. Skowron, J.P. Barthes, L. Jain, R. Sun, P. Morizet-Mahoudeaux, J. Liu, and N. Zhong. *Proceedings of The 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005)*. IEEE Computer Society. 2005. pp. 379-385.

[Pokahr et al. 2005c] A. Pokahr, L. Braubach, and W. Lamersdorf. *Jadex: A BDI Reasoning Engine*. R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Programing Multi-Agent Systems*. Kluwer Academic Publishers. 2005. pp.149-174.

[Rao and Georgeff 1995] A. Rao and M. Georgeff. *BDI Agents: from theory to practice*. V. Lesser. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*. The MIT Press. Cambridge, MA, USA. 1995. pp.312-319.

[Shoham 1993] Y. Shoham. *Agent-oriented programming*. D. G. Bobrow. *Artificial Intelligence Volume 60*. Elsevier. Amsterdam. 1993. pp.51-92.

[Winikoff 2005] M. Winikoff. *JACK Intelligent Agents: An Industrial Strength Platform*. R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Programing Multi-Agent Systems*. Kluwer Academic Publishers. 2005. pp.175-193.