ENGINEERING
INGEGNERIA
INFORMATICA

R&D Lab Palermo, Italy
http://res.eng.it

PRACTIcal reasONIng sySTem

PRACTIONIST Framework
User Guide

Table of contents

1 What is the PRACTIONIST framework?

PRACTIONIST (PRACTIcal reasONIng sySTem), is a new framework built on the Bratman's theory of practical reasoning to support the development of BDI agents in Java. The framework is built on top of JADE [11], a widespread platform that implements the FIPA specifications and that provides some core services, such as a communication infrastructure, agent life-cycle management, and so forth; therefore, in the PRACTIONIST framework, agents are deployed within JADE containers and have a belief base implemented in Prolog or in Java, as shown in the following figure:

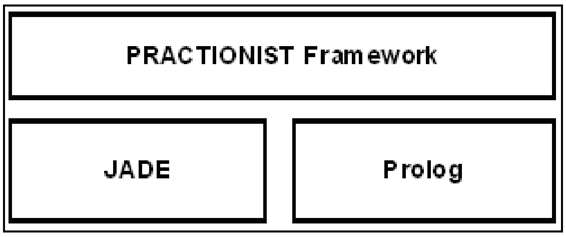


Figure 1-1 PRACTIONIST over JADE and Prolog.

The framework adopts a goal-oriented approach to develop BDI agents and stresses the separation between the deliberation process and the means-ends reasoning, with the abstraction of goal used to formally define both desires and intentions during the deliberation phase; in other words, PRACTIONIST agents can be programmed in terms of goals.

A PRACTIONIST agent is a software component endowed with the following elements (shown in Figure 1-2):

- a set of perceptions and the corresponding perceptors that listen to some relevant external stimuli;
- a set of beliefs representing the information the agent has got about both its internal state and the external environment;
- a set of goals the agent wishes or wants to pursue. They represent some states of affairs to bring about or activities to perform and will be related to either its desires or intentions (see below);
- a set of goal relations the agent uses during the deliberation process and means-ends reasoning;
- a set of plans that are the means to achieve its intentions;
- a set of actions the agent can perform to act over its environment; and
- a set of effectors that actually execute the actions.

As shown in Figure 1-2, PRACTIONIST agents are structured in two main layers: the framework defines the execution logic and provides the built-in components according to such logic, while the top layer includes the specific agent components to be implemented, in order to satisfy system requirements.

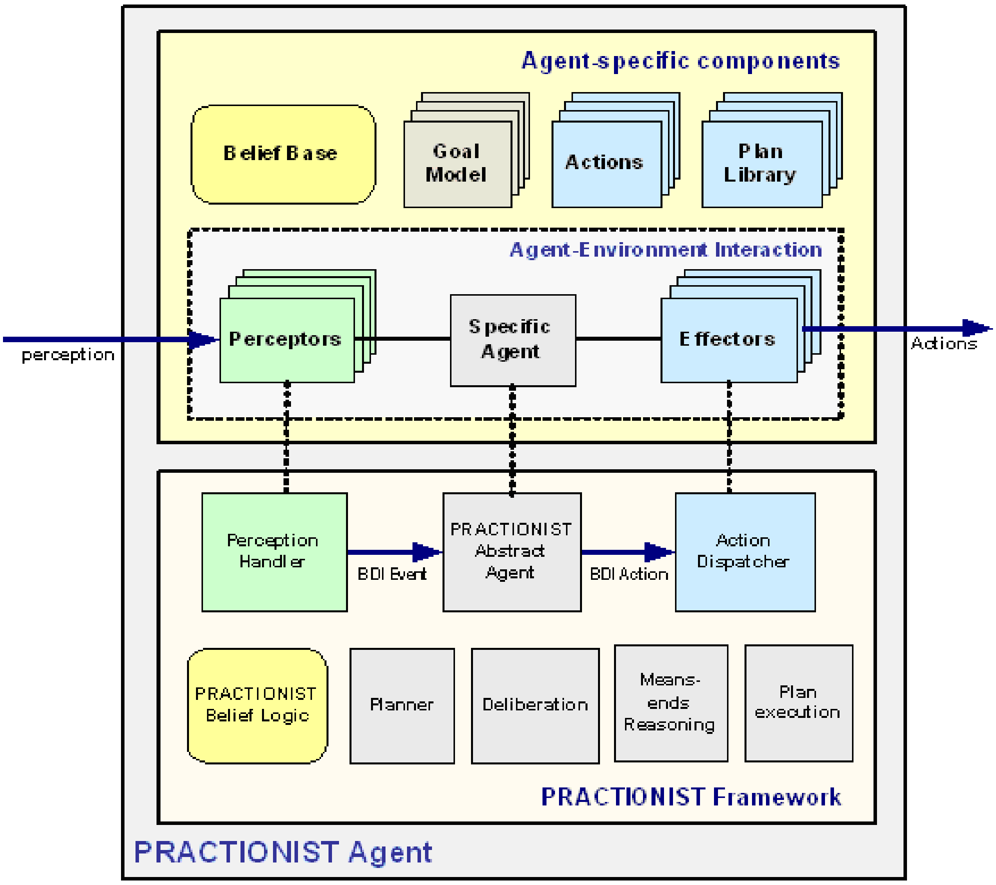


Figure 1-2 Components of PRACTIONIST agents.

Moreover, by using the Belief Logic, PRACTIONIST agents are able to reason about their beliefs and the other agent's beliefs, since beliefs are not simple grounded literals or data structures but modal logic formulas.

2 PRACTIONIST requirements

PRACTIONIST is a framework fully implemented in Java, so you need the Java Run Time Environment 1.4 or higher (http://java.sun.com/). Some others prerequisites of the PRACTIONIST framework are listed:

- JADE (http://jade.tilab.com/), a widespread platform that implements the FIPA specifications and provides some core services, such as a communication infrastructure, agent life-cycle management, and so forth. We have built and tested our framework with JADE 3.3.
- tuProlog (http://tuprolog.alice.unibo.it/), a Java-based Prolog which has been included in our framework with the version 1.3.0.
- JPL (http://www.swi-prolog.org/), a Java Interface to Prolog included in the SWI-Prolog distribution. The SWI-Prolog installation is required if agents you use in your applications have a prolog belief base. We have built and tested our framework with JPL 3.0.3 (it is included in the SWI-Prolog 5.4.7 executable file).
- log4j (http://logging.apache.org/), a logging framework included in the Apache Logging Services Project. We have tested our framework with log4j 1.2.8.
- Xerces (http://xerces.apache.org/), a Java XML Parser developed by the Apache Software Foundation. If agents you develop will use the choreography support builded in our framework, then it is mandatory the installation of Xerces. We have tested our framework with Xerces 1.1.

3 Installing PRACTIONIST

To install PRACTIONIST, you need to have the PRACTIONIST distribution file; in the download section of the PRACTIONIST web site (http://www.practionist.org), you can find the zip archive containing the framework already compiled.

You have to follow the steps below to install and use the PRACTIONIST framework in your projects:

1. download the zip file containing the PRACTIONIST jar archive;
2. download the JADE jar archives and the log4j jar archive;
3. download the jpl library (see the previous section for further details about the requirement of this library);
4. download the Xerces library (see the previous section for further details about the requirement of this library);
5. import the downloaded libraries into your project;

Remember to set the path of the imported jar archives into the class path to execute correctly your applications.

If you have already installed SWI-Prolog, you has to add the jpl.dll location to the PATH environment variable (e.g. %ProgramFiles%\pl\bin, if SWI-Prolog has been installed in the default directory).

It's strongly recommended to install Apache Ant (http://ant.apache.org/), as every executable component in the framework has an Ant build file associated to it; moreover it's necessary to add the Ant's bin location to the PATH environment variable (e.g. %ProgramFiles%\apache-ant-X.Y.Z\bin, where X.Y.Z denotes the Ant version installed).

4 Developing PRACTIONIST multi-agent systems

Once the PRACTIONIST framework is correctly installed, you can start developing your own software applications. The framework supports such a development phase by providing several useful libraries, including interfaces, abstract classes, default components, internal services implementing the computation model of PRACTIONIST agents, etc.

How to program PRACTIONIST agents is described in details in the PRACTIONIST Programmer Guide.

5 Starting PRACTIONIST agents

As stated in section 1, the PRACTIONIST framework is built on top of JADE. Therefore a running JADE platform represents a mandatory requirement to start a PRACTIONIST agent. We have defined a PRACTIONIST agent starter, which is a JADE agent with the purpose of starting a PRACTIONIST agent, initializing its main behaviour.

This agent is represented by the Java class `AgentStarter` included in the package `org.practionist.core` and requires some arguments to start a PRACTIONIST agent:

- the agent class, as the first parameter, which is a string representing the class name, including its package declaration (e.g. `myapp.agent.MyAgent`). This argument is mandatory: if it is missing then the agent initialization phase fails;
- `true` if you want to start the PRACTIONIST Agent Introspection Tool (PAIT), `false` otherwise;
- others arguments to pass to the agent in a string format, if there are.

You have different options to start your PRACTIONIST agent:

- you can define a “`build.xml`” file with the proper targets and use the Apache Ant tool to execute these;

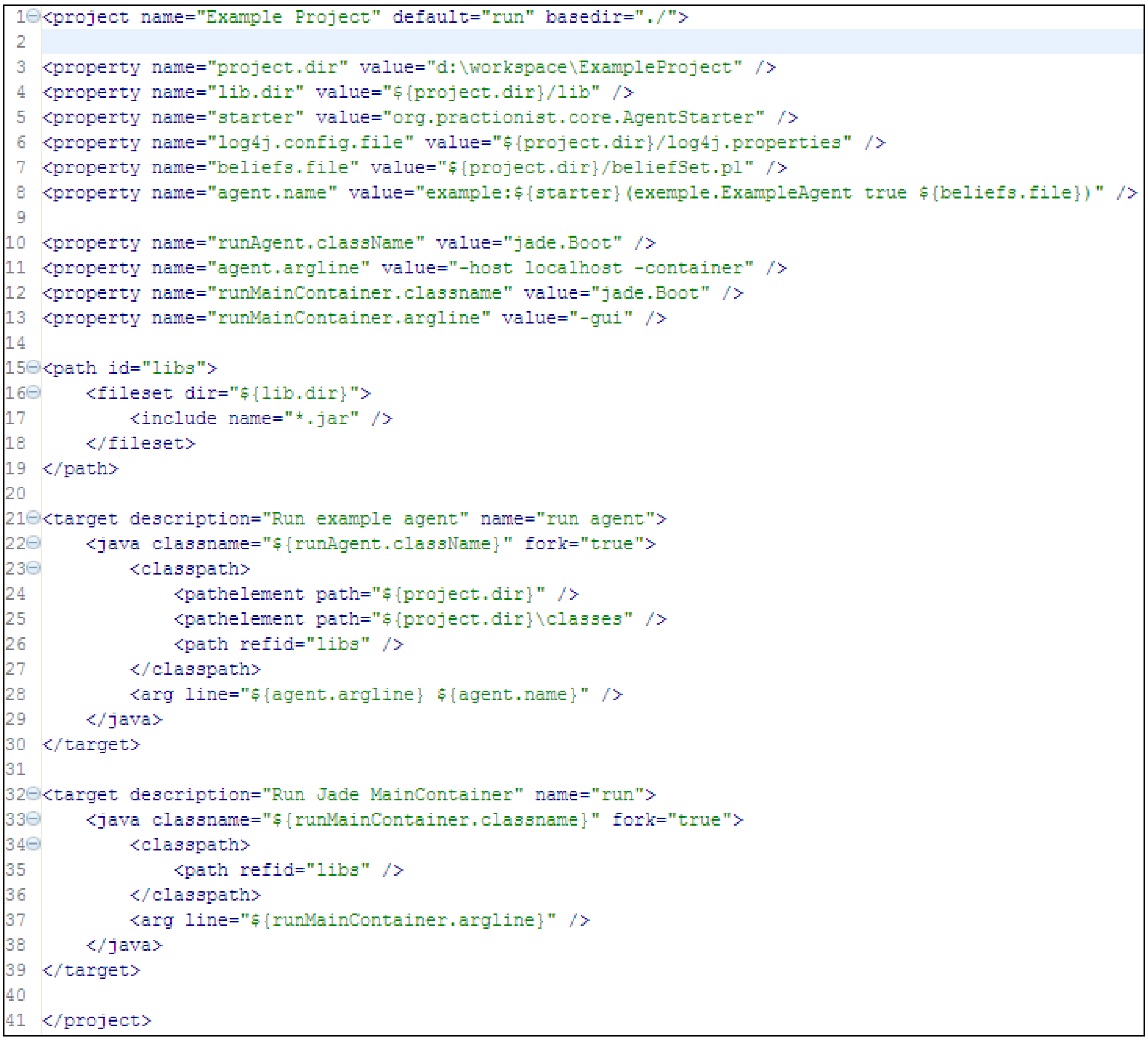
- you can use the PRACTIONIST Agent Starter GUI, by which you can set some parameters, such as the agent class, the agent name, etc., and finally
- you can use a batch file.

The following subsections describe how to start a PRACTIONIST agent by using each of the above methods.

5.1 Using the Ant tool

If you want start your agents by using the Apache Ant tool, then you have to create a build file with some targets, each one associated to an executable agent.

In the following figure, the build file required to execute the "example agent" is shown:



```xml
<project name="Example Project" default="run" basedir="./">

<property name="project.dir" value="d:\workspace\ExampleProject" />
<property name="lib.dir" value="${project.dir}/lib" />
<property name="starter" value="org.practionist.core.AgentStarter" />
<property name="log4j.config.file" value="${project.dir}/log4j.properties" />
<property name="beliefs.file" value="${project.dir}/beliefSet.pl" />
<property name="agent.name" value="example:${starter}(exemple.ExampleAgent true ${beliefs.file})" />

<property name="runAgent.className" value="jade.Boot" />
<property name="agent.argline" value="-host localhost -container" />
<property name="runMainContainer.classname" value="jade.Boot" />
<property name="runMainContainer.argline" value="-gui" />

<path id="libs">
    <fileset dir="${lib.dir}">
        <include name="*.jar" />
    </fileset>
</path>

<target description="Run example agent" name="run agent">
    <java classname="${runAgent.className}" fork="true">
        <classpath>
            <pathelement path="${project.dir}" />
            <pathelement path="${project.dir}\classes" />
            <path refid="libs" />
        </classpath>
        <arg line="${agent.argline} ${agent.name}" />
    </java>
</target>

<target description="Run Jade MainContainer" name="run">
    <java classname="${runMainContainer.classname}" fork="true">
        <classpath>
            <path refid="libs" />
        </classpath>
        <arg line="${runMainContainer.argline}" />
    </java>
</target>

</project>
```

Figure 5-1 An example of Ant build file.

You should focus your attention in the definition of the `agent.name` property: its value contains the name of the agent (e.g. `pippo`), the agent starter (`org.practionist.core.AgentStarter`), the PRACTIONIST agent class (`examples.ExampleAgent`) and its argument, that are the string "`true`" (to enable the PAIT tool) and the path of the file containing the initial belief base.

5.2 Using the Agent Starter GUI

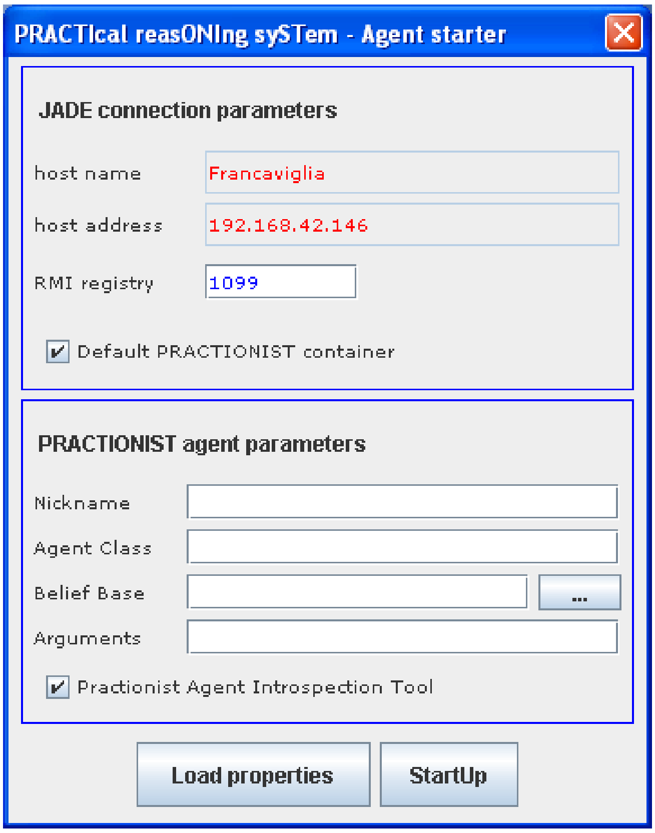
The class `AgentStarter` in the package `org.practionist.core` represents the GUI shown in Figure 5-2, which you can use to set or to load some properties required to start your agents.

In the upper part of the GUI, you can set the parameters regarding the JADE platform and container into which the agent has to be executed:

- The RMI Registry, that is an integer representing the port number where the Main Container is listening to container registrations; the default value is 1099. At the moment, the Main Container must be localized on a local JADE platform.
- The default PRACTIONIST container: if you select this check box, your agent will be created into a container called "PRACTIONIST", otherwise a new container will be created. Into the default container, only one agent with a prolog belief base at a time can be created (see the programmer's guide for more details).

Instead, in the lower part of the GUI, you have to set the parameters regarding the agent to execute:

- The nickname of the agent.
- The class identifying the agent.
- The file containing the initial belief base of the agent, if any.
- Some arguments the agent requires, if any.
- The PAIT tool, if you select this check box, the GUI of the PAIT tool regarding the agent will be created after the agent creation.



PRACTIcal reasONIng sySTem - Agent starter

JADE connection parameters

host name Francaviglia

host address 192.168.42.146

RMI registry 1099

☑ Default PRACTIONIST container

PRACTIONIST agent parameters

Nickname

Agent Class

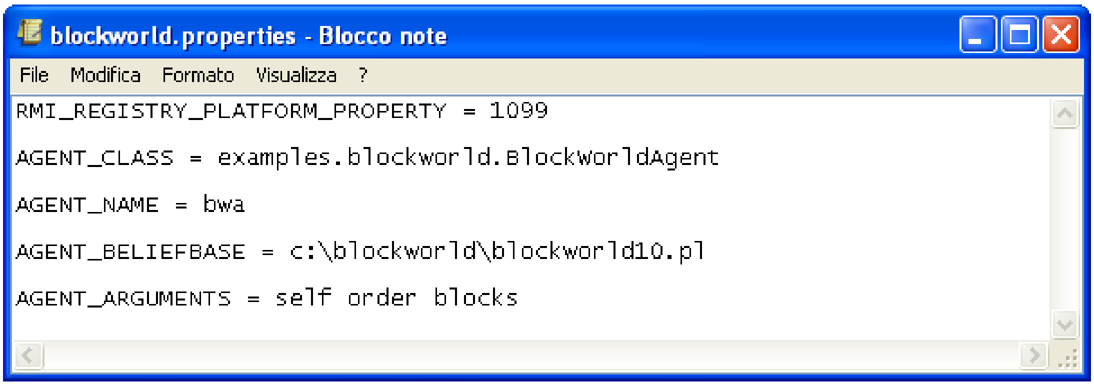
Belief Base

Arguments

☑ Practionist Agent Introspection Tool

Figure 5-2 PRACTIONIST Agent Starter GUI.

You can also set all these properties into a configuration file with the ".properties" extension, and load it by clicking on the "Load properties" button. An example of it is shown in the following figure:


blockworld.properties - Blocco note
File Modifica Formato Visualizza ?

```
RMI_REGISTRY_PLATFORM_PROPERTY = 1099

AGENT_CLASS = examples.blockworld.BlockWorldAgent

AGENT_NAME = bwa

AGENT_BELIEFBASE = c:\blockworld\blockworld10.pl

AGENT_ARGUMENTS = self order blocks
```

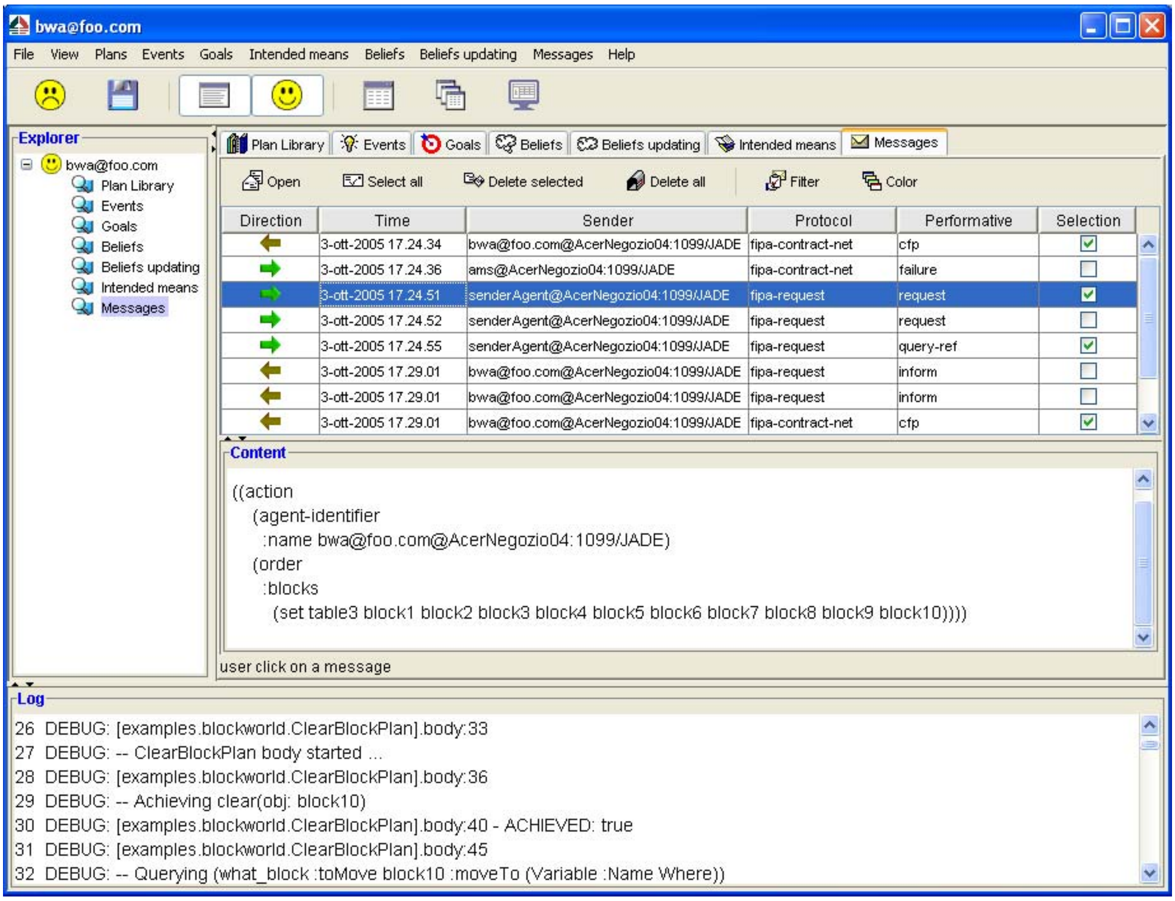
Figure 5-3 An example of configuration file.

Finally, you have to click on the “StartUp” button to start the agent.

6 Debugging a PRACTIONIST agent: the PAIT tool

The framework provides developers with the PRACTIONIST Agent Introspection Tool (PAIT), a visual integrated monitoring and debugging tool, which supports the analysis of the agent’s state during its execution. In particular, the PAIT can be suitable to display, test and debug the agents’ relevant entities and execution flow.

Each of these components can be observed at run-time through a set of specific tabs (see Figure 6-1); the content of each tab can be also displayed in an independent window.


bwa@foo.com
File View Plans Events Goals Intended means Beliefs Beliefs updating Messages Help
Explorer
bwa@foo.com
Plan Library
Events
Goals
Beliefs
Beliefs updating
Intended means
Messages
Plan Library Events Goals Beliefs Beliefs updating Intended means Messages
Open Select all Delete selected Delete all Filter Color
Content
user click on a message
Log

Direction	Time	Sender	Protocol	Performative	Selection
←	3-ott-2005 17.24.34	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-contract-net	cfp	☑
→	3-ott-2005 17.24.36	ams@AcerNegozio04:1099/JADE	fipa-contract-net	failure	☐
→	3-ott-2005 17.24.51	senderAgent@AcerNegozio04:1099/JADE	fipa-request	request	☑
→	3-ott-2005 17.24.52	senderAgent@AcerNegozio04:1099/JADE	fipa-request	request	☐
→	3-ott-2005 17.24.55	senderAgent@AcerNegozio04:1099/JADE	fipa-request	query-ref	☑
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-request	inform	☐
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-request	inform	☐
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-contract-net	cfp	☑

```
((action
   (agent-identifier
     :name bwa@foo.com@AcerNegozio04:1099/JADE)
   (order
     :blocks
       (set table3 block1 block2 block3 block4 block5 block6 block7 block8 block9 block10))))
```

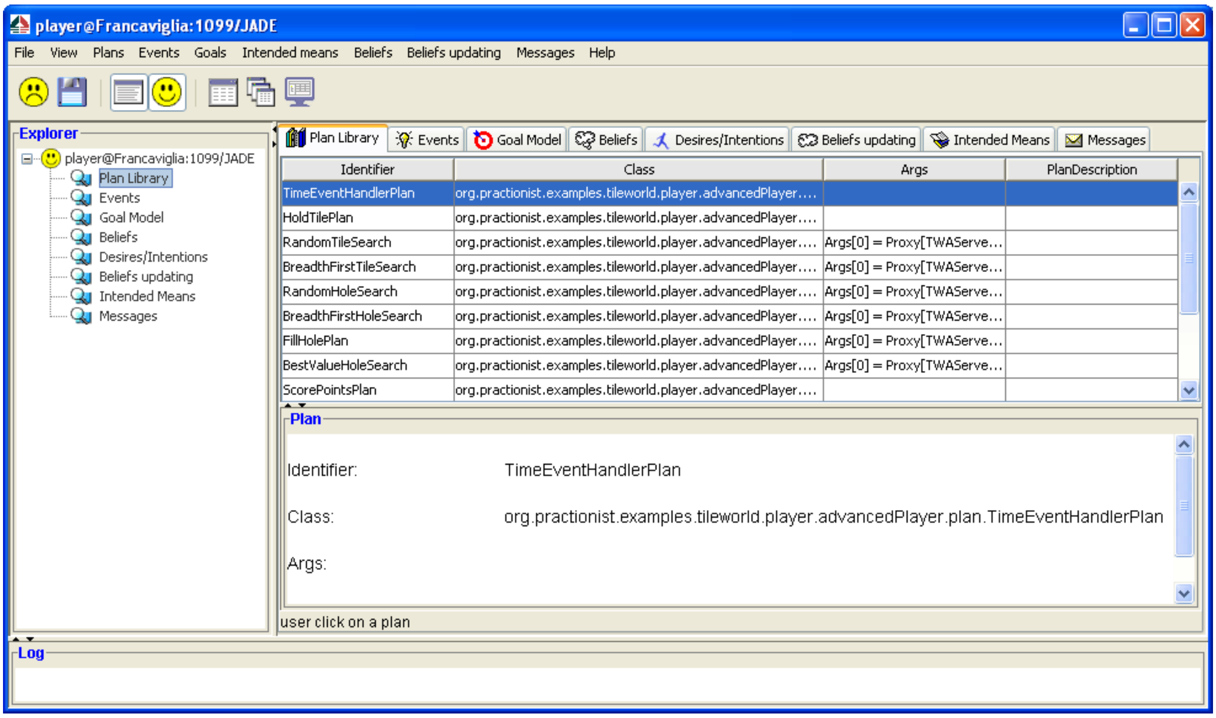
```
26  DEBUG: [examples.blockworld.ClearBlockPlan].body:33
27  DEBUG: -- ClearBlockPlan body started ...
28  DEBUG: [examples.blockworld.ClearBlockPlan].body:36
29  DEBUG: -- Achieving clear(obj: block10)
30  DEBUG: [examples.blockworld.ClearBlockPlan].body:40 - ACHIEVED: true
31  DEBUG: [examples.blockworld.ClearBlockPlan].body:45
32  DEBUG: -- Querying (what_block :toMove block10 :moveTo (Variable :Name Where))
```

Figure 6-1 The PRACTIONIST Agent Introspection Tool (PAIT).

In the following subsections, the views provided by PAIT are presented.

6.1 Plan Library

This view shows the list of plans within the plan library of the considered agent (Figure 6-2). Some of these plans may have an associated plan description, that can be displayed as in Figure 6-3.


player@Francaviglia:1099/JADE
File View Plans Events Goals Intended means Beliefs Beliefs updating Messages Help
Explorer
player@Francaviglia:1099/JADE
Plan Library
Events
Goal Model
Beliefs
Desires/Intentions
Beliefs updating
Intended Means
Messages
Plan Library Events Goal Model Beliefs Desires/Intentions Beliefs updating Intended Means Messages
Plan
Log

Identifier	Class	Args	PlanDescription
TimeEventHandlerPlan	org.practionist.examples.tileworld.player.advancedPlayer....		
HoldTilePlan	org.practionist.examples.tileworld.player.advancedPlayer....		
RandomTileSearch	org.practionist.examples.tileworld.player.advancedPlayer....	Args[0] = Proxy[TWAServe...	
BreadthFirstTileSearch	org.practionist.examples.tileworld.player.advancedPlayer....	Args[0] = Proxy[TWAServe...	
RandomHoleSearch	org.practionist.examples.tileworld.player.advancedPlayer....	Args[0] = Proxy[TWAServe...	
BreadthFirstHoleSearch	org.practionist.examples.tileworld.player.advancedPlayer....	Args[0] = Proxy[TWAServe...	
FillHolePlan	org.practionist.examples.tileworld.player.advancedPlayer....	Args[0] = Proxy[TWAServe...	
BestValueHoleSearch	org.practionist.examples.tileworld.player.advancedPlayer....	Args[0] = Proxy[TWAServe...	
ScorePointsPlan	org.practionist.examples.tileworld.player.advancedPlayer....		

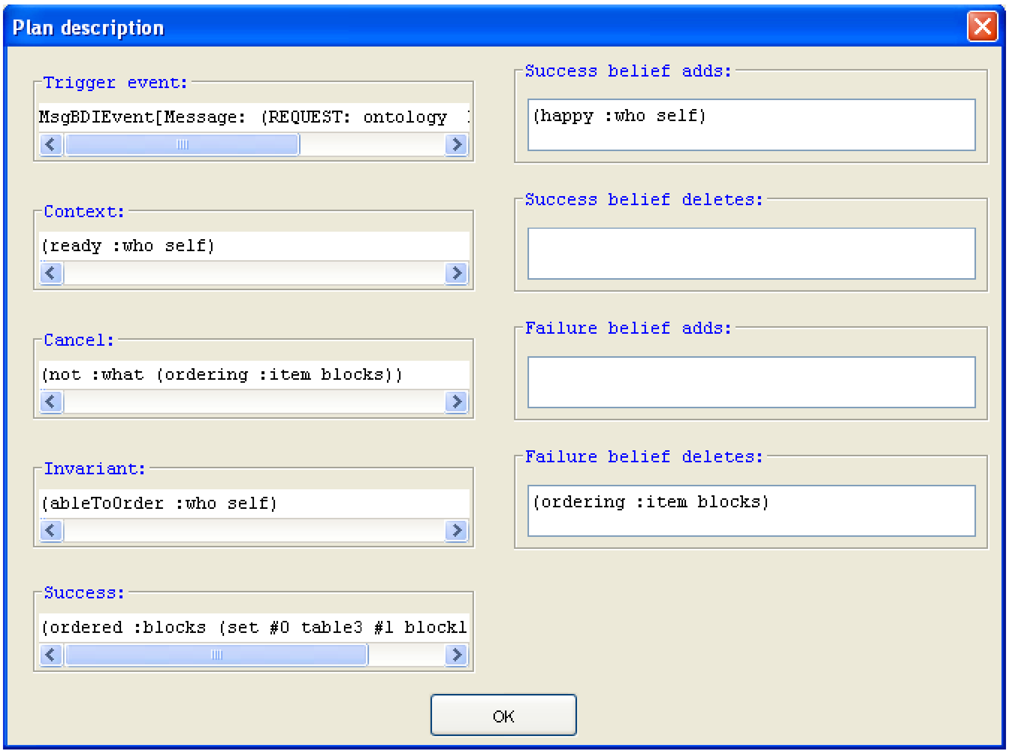
Identifier: TimeEventHandlerPlan

Class: org.practionist.examples.tileworld.player.advancedPlayer.plan.TimeEventHandlerPlan

Args:

user click on a plan

Figure 6-2 Plan Library view.



Trigger event: MsgBDIEvent[Message: (REQUEST: ontology

Context: (ready :who self)

Cancel: (not :what (ordering :item blocks))

Invariant: (ableToOrder :who self)

Success: (ordered :blocks (set #0 table3 #1 block1

Success belief adds: (happy :who self)

Success belief deletes:

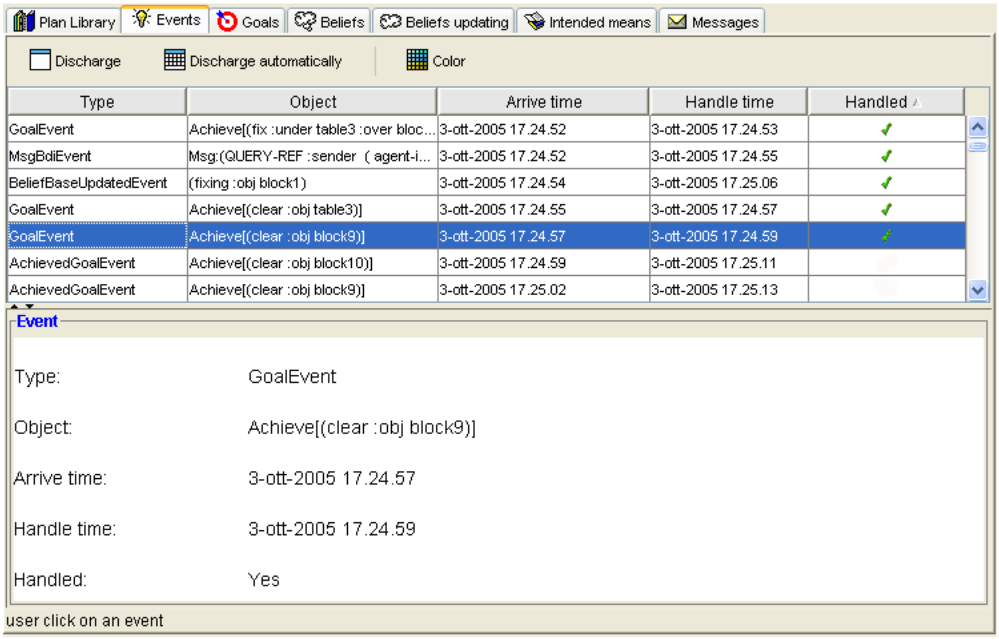
Failure belief adds:

Failure belief deletes: (ordering :item blocks)

Figure 6-3 Plan description.

6.2 Events

This view shows the list of events (i.e. desired goals, perceptions, changes in its beliefs) that the considered agent can handle (Figure 6-4).



Type	Object	Arrive time	Handle time	Handled
GoalEvent	Achieve[(fix :under table3 :over bloc...	3-ott-2005 17.24.52	3-ott-2005 17.24.53	✓
MsgBdiEvent	Msg:(QUERY-REF :sender (agent-i...	3-ott-2005 17.24.52	3-ott-2005 17.24.55	✓
BeliefBaseUpdatedEvent	(fixing :obj block1)	3-ott-2005 17.24.54	3-ott-2005 17.25.06	✓
GoalEvent	Achieve[(clear :obj table3)]	3-ott-2005 17.24.55	3-ott-2005 17.24.57	✓
GoalEvent	Achieve[(clear :obj block9)]	3-ott-2005 17.24.57	3-ott-2005 17.24.59	✓
AchievedGoalEvent	Achieve[(clear :obj block10)]	3-ott-2005 17.24.59	3-ott-2005 17.25.11	
AchievedGoalEvent	Achieve[(clear :obj block9)]	3-ott-2005 17.25.02	3-ott-2005 17.25.13	

Event

Type: GoalEvent

Object: Achieve[(clear :obj block9)]

Arrive time: 3-ott-2005 17.24.57

Handle time: 3-ott-2005 17.24.59

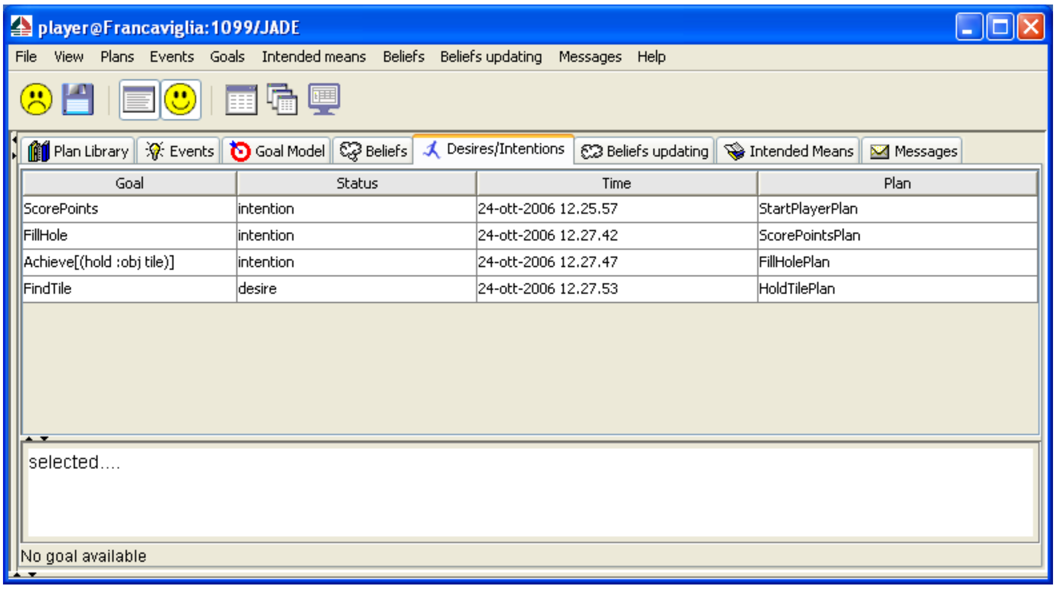
Handled: Yes

user click on an event

Figure 6-4 Events view.

6.3 Desires/Intentions

This view shows the list of current desires and intentions of the considered agent (Figure 6-5).



Goal	Status	Time	Plan
ScorePoints	intention	24-ott-2006 12.25.57	StartPlayerPlan
FillHole	intention	24-ott-2006 12.27.42	ScorePointsPlan
Achieve[(hold :obj tile)]	intention	24-ott-2006 12.27.47	FillHolePlan
FindTile	desire	24-ott-2006 12.27.53	HoldTilePlan

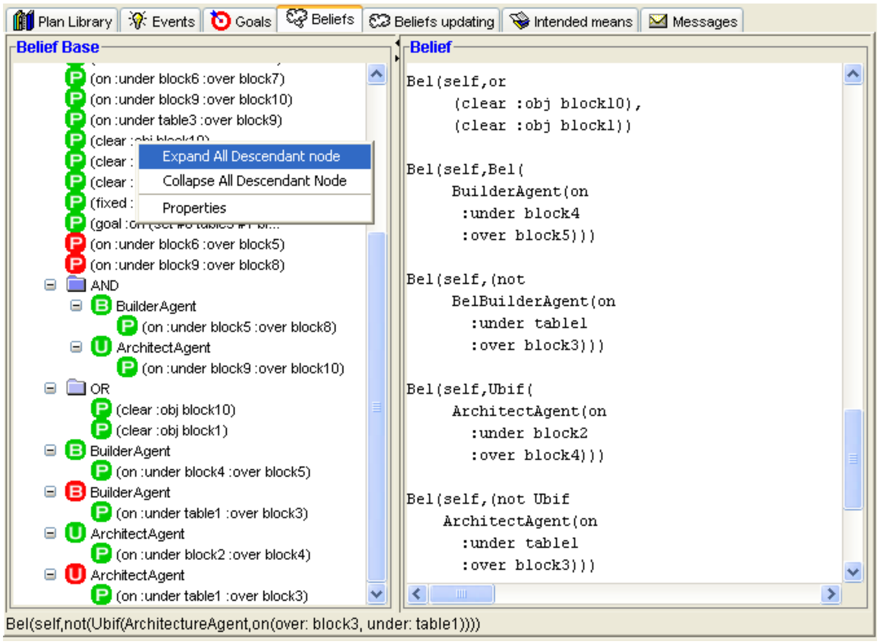
selected....

No goal available

Figure 6-5 Desires/Intentions view.

6.4 Beliefs

This view shows the whole belief base of the considered agent, providing the opportunity to browse it by using the tree structure on the left.



```
P (on :under block6 :over block7)
P (on :under block9 :over block10)
P (on :under table3 :over block9)
P (clear :obj block10)
P (clear :
P (clear :
P (fixed :
P (goal :on (set #6 table3 #1 bl...
P (on :under block6 :over block5)
P (on :under block9 :over block8)
⊟ AND
   ⊟ B BuilderAgent
        P (on :under block5 :over block8)
   ⊟ U ArchitectAgent
        P (on :under block9 :over block10)
⊟ OR
     P (clear :obj block10)
     P (clear :obj block1)
⊟ B BuilderAgent
     P (on :under block4 :over block5)
⊟ B BuilderAgent
     P (on :under table1 :over block3)
⊟ U ArchitectAgent
     P (on :under block2 :over block4)
⊟ U ArchitectAgent
     P (on :under table1 :over block3)
```

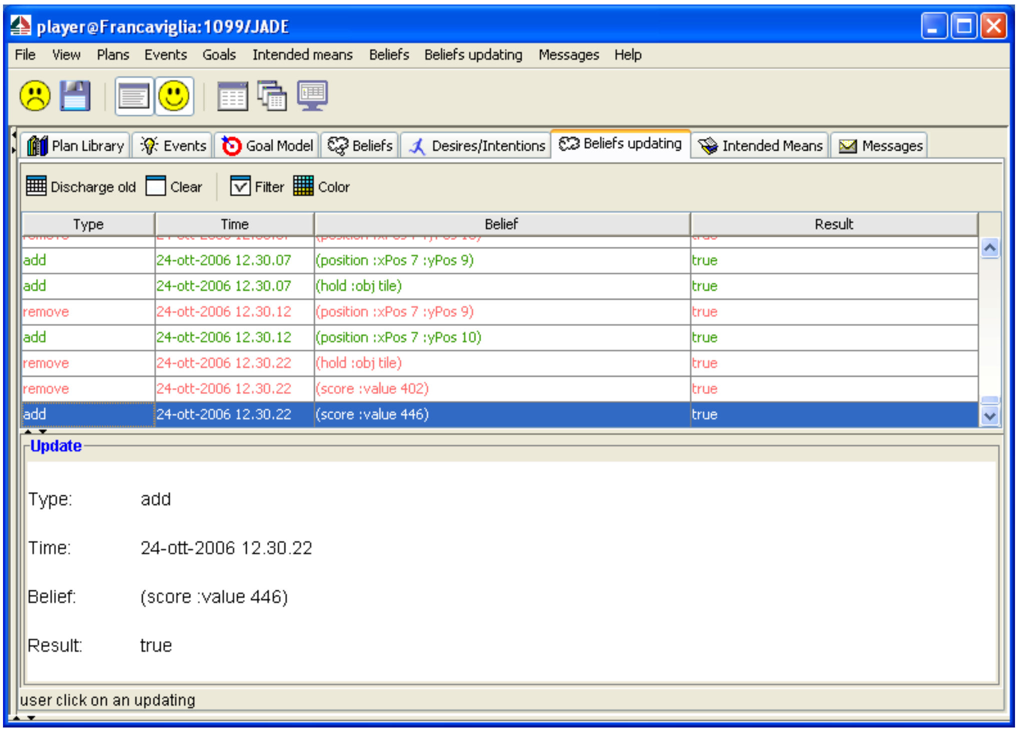
```
Bel(self,or
     (clear :obj block10),
     (clear :obj block1))

Bel(self,Bel(
     BuilderAgent(on
      :under block4
      :over block5)))

Bel(self,(not
     BelBuilderAgent(on
       :under table1
       :over block3)))

Bel(self,Ubif(
     ArchitectAgent(on
       :under block2
       :over block4)))

Bel(self,(not Ubif
    ArchitectAgent(on
      :under table1
      :over block3)))
```

Bel(self,not(Ubif(ArchitectureAgent,on(over: block3, under: table1))))

Figure 6-6 Beliefs view.

6.5 Beliefs updates

This view shows the list of beliefs updates within the considered agent (Figure 6-7).



☐ Clear ☑ Filter

Type	Time	Belief	Result
add	24-ott-2006 12.30.07	(position :xPos 7 :yPos 9)	true
add	24-ott-2006 12.30.07	(hold :obj tile)	true
remove	24-ott-2006 12.30.12	(position :xPos 7 :yPos 9)	true
add	24-ott-2006 12.30.12	(position :xPos 7 :yPos 10)	true
remove	24-ott-2006 12.30.22	(hold :obj tile)	true
remove	24-ott-2006 12.30.22	(score :value 402)	true
add	24-ott-2006 12.30.22	(score :value 446)	true

Type: add

Time: 24-ott-2006 12.30.22

Belief: (score :value 446)

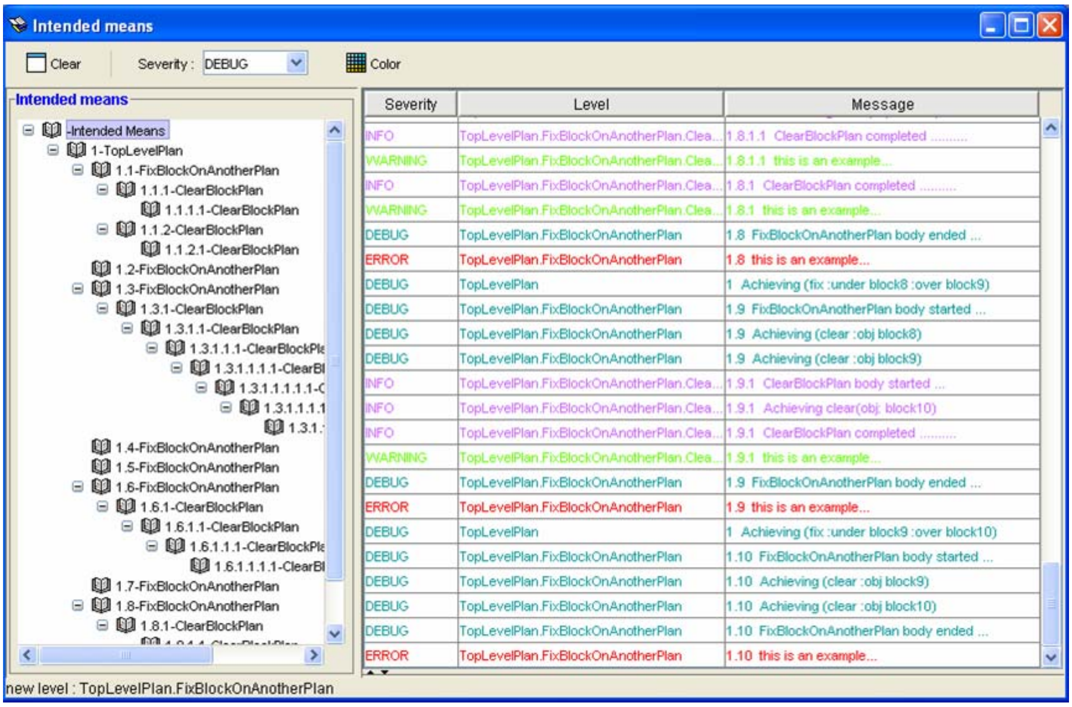
Result: true

user click on an updating

Figure 6-7 Belief updates view.

6.6 Intended means

This view shows the structure of intended means of the considered agent (Figure 6-8). On the left panel, the nested structure of the intended means is reported, while the main panel includes the traced messages within the selected intended means and its upper intended means.



Severity : DEBUG

```
⊟ -Intended Means
   ⊟ 1-TopLevelPlan
      ⊟ 1.1-FixBlockOnAnotherPlan
         ⊟ 1.1.1-ClearBlockPlan
               1.1.1.1-ClearBlockPlan
         ⊟ 1.1.2-ClearBlockPlan
               1.1.2.1-ClearBlockPlan
         1.2-FixBlockOnAnotherPlan
      ⊟ 1.3-FixBlockOnAnotherPlan
         ⊟ 1.3.1-ClearBlockPlan
            ⊟ 1.3.1.1-ClearBlockPlan
               ⊟ 1.3.1.1.1-ClearBlockPl
                  ⊟ 1.3.1.1.1.1-ClearBl
                     ⊟ 1.3.1.1.1.1.1-C
                        ⊟ 1.3.1.1.1.1
                              1.3.1.
         1.4-FixBlockOnAnotherPlan
         1.5-FixBlockOnAnotherPlan
      ⊟ 1.6-FixBlockOnAnotherPlan
         ⊟ 1.6.1-ClearBlockPlan
            ⊟ 1.6.1.1-ClearBlockPlan
               ⊟ 1.6.1.1.1-ClearBlockPla
                     1.6.1.1.1.1-ClearBl
         1.7-FixBlockOnAnotherPlan
      ⊟ 1.8-FixBlockOnAnotherPlan
         ⊟ 1.8.1-ClearBlockPlan
```

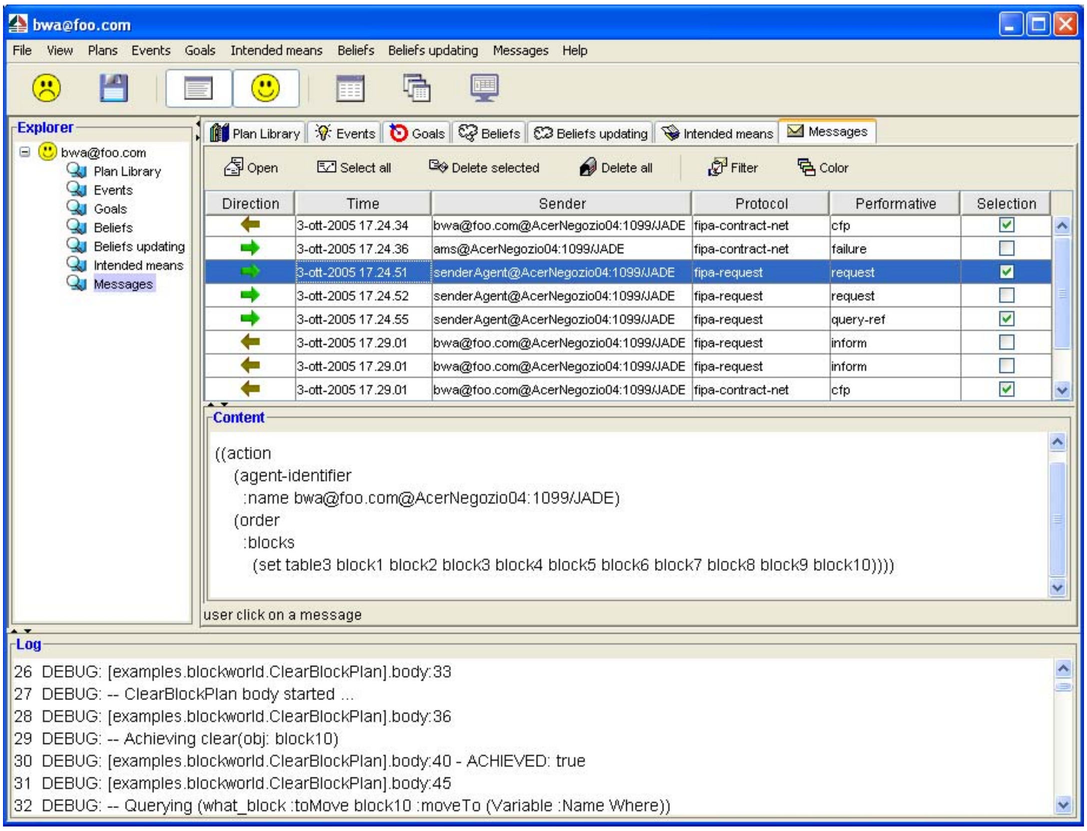
Severity	Level	Message
INFO	TopLevelPlan.FixBlockOnAnotherPlan.Clea...	1.8.1.1 ClearBlockPlan completed ....
WARNING	TopLevelPlan.FixBlockOnAnotherPlan.Clea...	1.8.1.1 this is an example...
INFO	TopLevelPlan.FixBlockOnAnotherPlan.Clea...	1.8.1 ClearBlockPlan completed ....
WARNING	TopLevelPlan.FixBlockOnAnotherPlan.Clea...	1.8.1 this is an example...
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.8 FixBlockOnAnotherPlan body ended ...
ERROR	TopLevelPlan.FixBlockOnAnotherPlan	1.8 this is an example...
DEBUG	TopLevelPlan	1 Achieving (fix :under block8 :over block9)
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.9 FixBlockOnAnotherPlan body started ...
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.9 Achieving (clear :obj block8)
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.9 Achieving (clear :obj block9)
INFO	TopLevelPlan.FixBlockOnAnotherPlan.Clea...	1.9.1 ClearBlockPlan body started ...
INFO	TopLevelPlan.FixBlockOnAnotherPlan.Clea...	1.9.1 Achieving clear(obj: block10)
INFO	TopLevelPlan.FixBlockOnAnotherPlan.Clea...	1.9.1 ClearBlockPlan completed ....
WARNING	TopLevelPlan.FixBlockOnAnotherPlan.Clea...	1.9.1 this is an example...
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.9 FixBlockOnAnotherPlan body ended ...
ERROR	TopLevelPlan.FixBlockOnAnotherPlan	1.9 this is an example...
DEBUG	TopLevelPlan	1 Achieving (fix :under block9 :over block10)
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.10 FixBlockOnAnotherPlan body started ...
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.10 Achieving (clear :obj block9)
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.10 Achieving (clear :obj block10)
DEBUG	TopLevelPlan.FixBlockOnAnotherPlan	1.10 FixBlockOnAnotherPlan body ended ...
ERROR	TopLevelPlan.FixBlockOnAnotherPlan	1.10 this is an example...

new level : TopLevelPlan.FixBlockOnAnotherPlan

Figure 6-8 Intended means view.

6.7 Messages

This view shows the list of messages sent and received by the considered agent (Figure 6-9).


bwa@foo.com
File View Plans Events Goals Intended means Beliefs Beliefs updating Messages Help
Explorer
bwa@foo.com
Plan Library
Events
Goals
Beliefs
Beliefs updating
Intended means
Messages
Plan Library Events Goals Beliefs Beliefs updating Intended means Messages
Open Select all Delete selected Delete all Filter Color

Direction	Time	Sender	Protocol	Performative	Selection
⬅	3-ott-2005 17.24.34	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-contract-net	cfp	☑
➡	3-ott-2005 17.24.36	ams@AcerNegozio04:1099/JADE	fipa-contract-net	failure	□
➡	3-ott-2005 17.24.51	senderAgent@AcerNegozio04:1099/JADE	fipa-request	request	☑
➡	3-ott-2005 17.24.52	senderAgent@AcerNegozio04:1099/JADE	fipa-request	request	□
➡	3-ott-2005 17.24.55	senderAgent@AcerNegozio04:1099/JADE	fipa-request	query-ref	☑
⬅	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-request	inform	□
⬅	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-request	inform	□
⬅	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-contract-net	cfp	☑

Content

```
((action
   (agent-identifier
     :name bwa@foo.com@AcerNegozio04:1099/JADE)
   (order
     :blocks
       (set table3 block1 block2 block3 block4 block5 block6 block7 block8 block9 block10))))
```

user click on a message

Log

```
26  DEBUG: [examples.blockworld.ClearBlockPlan].body:33
27  DEBUG: -- ClearBlockPlan body started ...
28  DEBUG: [examples.blockworld.ClearBlockPlan].body:36
29  DEBUG: -- Achieving clear(obj: block10)
30  DEBUG: [examples.blockworld.ClearBlockPlan].body:40 - ACHIEVED: true
31  DEBUG: [examples.blockworld.ClearBlockPlan].body:45
32  DEBUG: -- Querying (what_block :toMove block10 :moveTo (Variable :Name Where))
```

Figure 6-9 Messages view.