Architecture of Java-based Agent Frameworks

W.Pasman, 6 may 2008

Introduction

This document investigates the architecture choices made in various Java-based agent frameworks and middleware. The focus in this document is on the multi-agent, debugging and modularization questions. Particularly an attempt was made to answer the following questions:

- What are the agents: separate java thread, java process, system process, or a function call by a platform scheduler?
- Are the agents autonomous, to which extent?
- What is communication model between agents? Is communication just another 'action' to the environment, or special thing?
- How is the modularization, and what is the core functionality? Is there a separate runtime core?
- Debug options: tracing, resetting, stepping? How is resetting (per agent, system wide) organized?
- Are there special scheduling/execution models (at core)?
- Who launches new agents, and who initializes them?
- Is there a syntax directed editor in the platform to edit the agents?
- Is there MAS set-up support?
- Is there an IDE? How is the relation with the runtime core?

A number of frameworks only provides means for communication and some facilitating services like yellow pages. We call such frameworks 'middleware'. We will discuss the JADE and the Saci middleware. Other frameworks make more detailed restrictions to the agents that can be used and methods to define an environment, for instance they may have a separate programming language to create an agent or a multi-agent system (**MAS**). Actually this distinction is very thin, JADE agents are written in Java but they can not just be any Java thread, but they must extend jade.Agent class.
Scheduling is about which agent(s) get(s) the CPU(s) and when. Of course a specific way of scheduling can always be created by having the agents cooperate and using a set of messages to synchronise them as required. Scheduling becomes more relevant in *absence* of such a cooperation scheme. In that case the core may still be able to estimate a good scheduling in order to complete the tasks of all agents as quick as possible.
Finally we give an overview of a number of key differences.

Terminology

The various agent platforms use different terminology for comparible services. Table 1 shows a description of the various tools and modules, and the names associated with them.

Table 1. Description of required tools and modules, and names used for that in various agent systems.

Description	Names
Tool that shows messages passed through the system	Sniffer, (Message) Tracer, Communication Monitor
Tool showing the internal state of an agent	Introspector, Mind Inspector, State Tracer, Agent Inspector
Tool enabling configuration of the MAS	Platform Controller, Core Controller GUI, Remote Agent Management Unit, Launcher, Admin, Infrastructure Controller, MAS Console, Control Center, Client
Software providing basic functionality for a MAS of a certain type[1]	Core, Runtime Multi-agent Platform, Infrastructure, Agent Management System
Tool enabling editing and running a MAS	Development Environment , IDE, Central Controller
Tool enabling to halt, step and run the core and/or individual agents	Debugger, Agent Tracing Controller
Software that allows access to a database of service names with associated agents delivering those services	Directory Facilitator, Name Server

[1] If the core is for instance to run JACK agents the core would be of the 'JACK' type.

1. Middleware

Middleware is a software layer offering basic functionality to implement distributed multi agent systems, such as launching and killing agents, and communication. We will investigate two versions of middleware in more detail:

JADE (Java Agent DEvelopment Framework) [JADE] is middleware to simplify the implementation of multi-agent systems. It complies with the FIPA specifications and offsers a set of graphical tools to support debugging and deployment phases.

Saci (Simple Agent Communication Infrastructure) [Saci] is a Java API and set of tools supporting communication between distributed agents. This is middleware similar to JADE, but without debugging tools.

1.1 JADE

JADE is middleware enabling agent management and communication. Following the FIPA specification, Jade agents are autonomous, proactive and social entities. Communication between agents runs via asynchronous messages.
Each agent in JADE is running in its own (single) Java thread. The JADE system enables every Serializable agent to be halted, moved to another container and resume execution. A container can be just another thread within the same JVM, another JVM on the same machine, or even a JVM on another machine.
In practice, there are limits to the 'autonomy' of an agent. First, the OS on which a JVM runs can always kill a JVM. Second, agents can request the AMS (Agent Management System) to kill or move other agents. There are ideas about regulating this but this seems still under development.
To my knowledge there is no system-wide scheduling model. All agents are individual threads, and the Java scheduler will distribute the available processor cycles (which in turn is determined by the scheduler of the particular OS where Java is being run on).
New agents can be created by the AMS. Agents initialise themselves. The AMS is not expected to manipulate the agent's internals directly, it can only request actions, and the agent can only be killed or suspended by the AMS.
There is no agent development kit, nor a MAS development kit, in JADE itself. But there are numerous agent environments that use JADE as a basis.

Tracing, Debugging

Figure 1 shows the introspector window of JADE. At the core, the agent is just another JAVA program and the introspector uses the JVM to control the agent state.

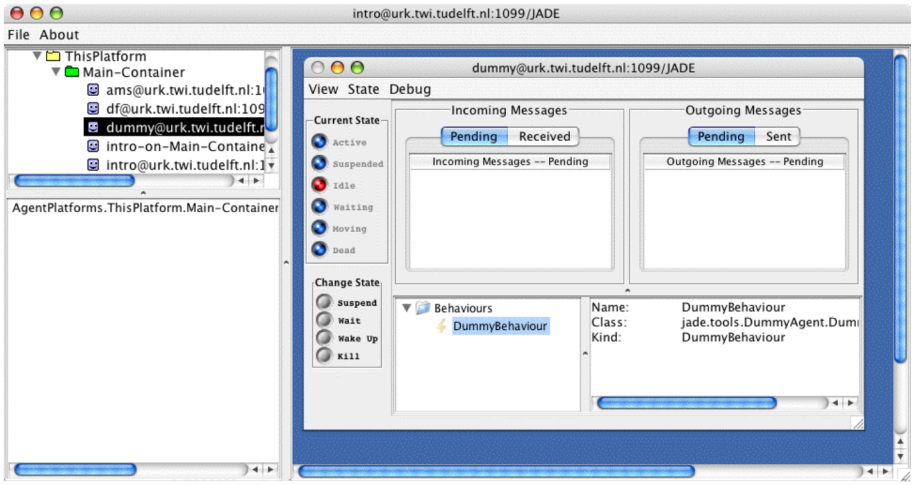

Figure 1. Introspector window of JADE.

The agent can be put into "step" debugging mode, by using the Debug menu (Figure 2). Stepping does not seem to do anything but break will suspend the agent when a behaviourchangestate event occurs, and slow will make the agent sleep for 0.5s in such a case. From inspection of the code, it appears that the Agent Management System (**AMS**, the 'platform manager') provides debug hooks, as follows. An agent (e.g. the Introspector) can request the AMS to debug a given agent. The AMS will then send information about all state change events of given agent, and wait for a reply before the given agent can continue. In the Agent code, all relevant changes are expected to be registered via notifyChangeBehaviour-State (and other similar functions). The notifyXXX functions are part of the AgentToolkit interface, and can be set –even while running– with the setToolkit function that is part of every agent. This allows the AMS to plug in a debug handler if needed[2].

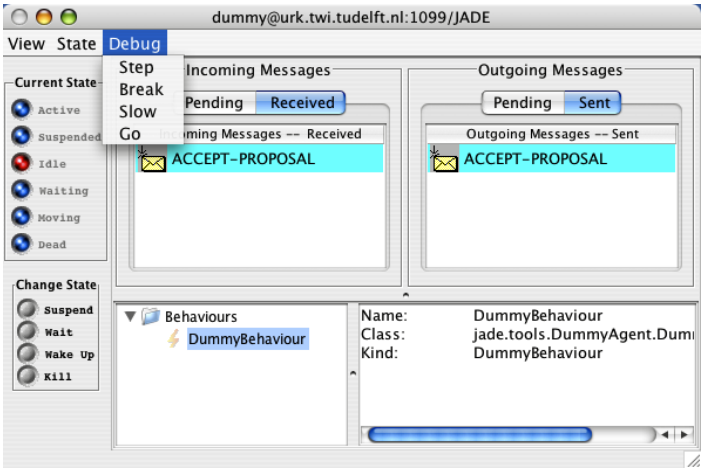

Figure 2. Debug options for an agent.

On a higher level, messages between agents can be traced with the sniffer (Figure 3). JADE has a top level platform controller with a GUI shown in Figure 4.
The manual indicates that all JADE agents have reset functions, and that they may be called whenever an agent crashes, to reinitialize it. However there seems no way to call them externally from for instance the AMS. The only way to reset an agent seems to have the agent accept a special reset message, and letting the agent reset whenever such a message is received. JADE does not seem to provide a platform-wide reset.

[2] Actually the AgentContainer plugs in a toolkit into the agent, which uses the CommandProcessor to filter kernel call.

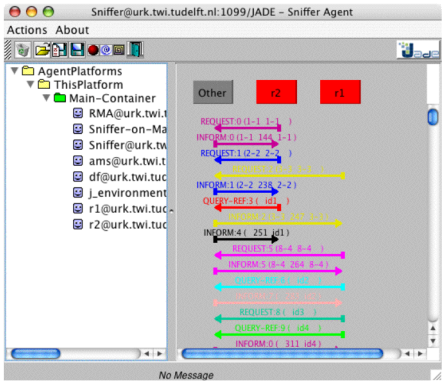

Figure 3. JADE Sniffer to trace messages.

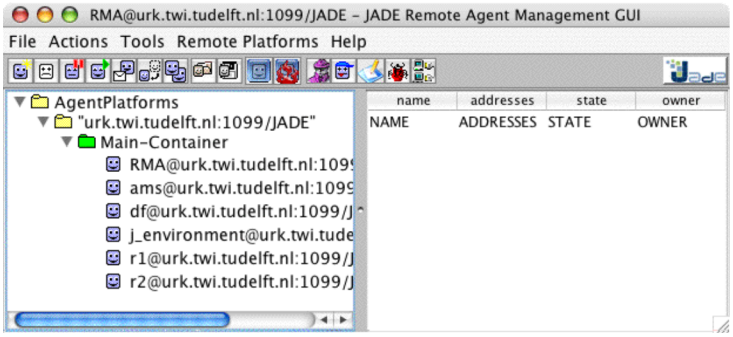

Figure 4. The JADE Platform Controller (RMA) GUI.

Modularization

JADE consists of three .jar files: jade.jar, jadeTools.jar, and http.jar, with optionally the iiop.jar (Internet Inter-ORB Protocol where ORB stands for Object Request Broker). The jade.jar contains the services for the AMS, containers, messaging, and interface definitions. These services are all without GUI. GUIs for these services are all found in the jadeTools, and these tools use messages to communicate with the services.

GUI Design

The Introspector (Figure 1) :

- I like the possibility to open multiple debugger windows, one for each agent.
- note that this window is conflicting with the Apple UI design guidelines [apple07], for instance you have to control-click on an agent in a container to get debugging for it, and the debugger window appears in a weird looking way inside the introspector window.
- Opening a window for an agent with right-clicking on agents, a very unintuitive and unexpected action for Macintosh users.
- The buttons in the left column of an agent introspector are fancy but non-standard Java. It is not clear which ones are pressable and which not. Only the text besides them is indicative that the higher ones are indicative only, while the lower 'change state' buttons are clickable. Clicking on the lower ones does not do anything though.
- One has to click on the "Debug" menu of this sub-window to actually change the state. This is also not intuitive for Macintosh users, menu items are selected from the top of the screen, not from a fake menu bar inside a window.
- The Introspector could have been used also to collect agent-specific messages. As it is now simple debug messages are just dumped to the console.
- For me this introspector does not work. It does not show sufficient internals of the agent to do debugging. To debug Jade agents usefully one needs stepping through Java code and the introspector does not offer that.

For the RMA GUI (Figure 4),

- The agent platform actions are available here as expected. Agents can be loaded, started, killed and suspended. There is quite some powerful functionality: agents can even be cloned, saved, and moved (all while they are running!). The "send message to agent" button is good to have but seems of another league.
- I like the availability of all tools in a single bar. The sniffer, dummyagent and debugger icons are natural but generally icons for actions are useless for me (e.g. what does a smiling agent with a ">" in the top right mean, or a smiling agent behind an envelop?). Custom icons should be used only if they are immediately recognizable or meaningful [UXG, p.136]. If you hover the mouse over the icons there does not even appear an explanation, making these icons completely useless for me. Luckily you can still access

all functionality via the window menu or via right-clicking (not Macintosh style but it works).

- Again we have most mentioned GUI design issues as mentioned for the Introspector.
- You can not edit agents from this central controller, nor with one of the tools. I think the JADE people would see such an editor as an optional extra tool that might be startable from the RMA, for instance with another button next to the debugger.
- The right half of the RMA window seems not too useful, here you can view the name, address, state and owner of agents that are selected in the left column.

1.2 Saci

In Saci, agents communicate with KQML messages.
Similar to JADE, Saci offers also facilities to create, kill and move agents (Figure 5), a message tracer (Figure 6) and a yellow page service. In contrast with JADE, Saci does not seem to support debug options like halting an agent or introspection of agent behaviour. Saci also does not offer editors.

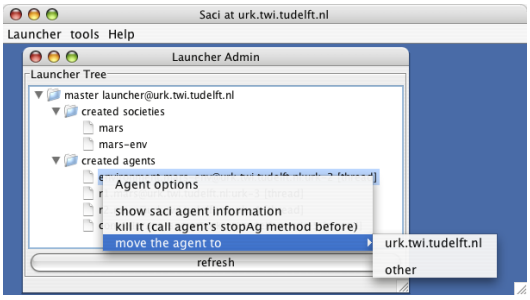

Figure 5. Saci launcher GUI.

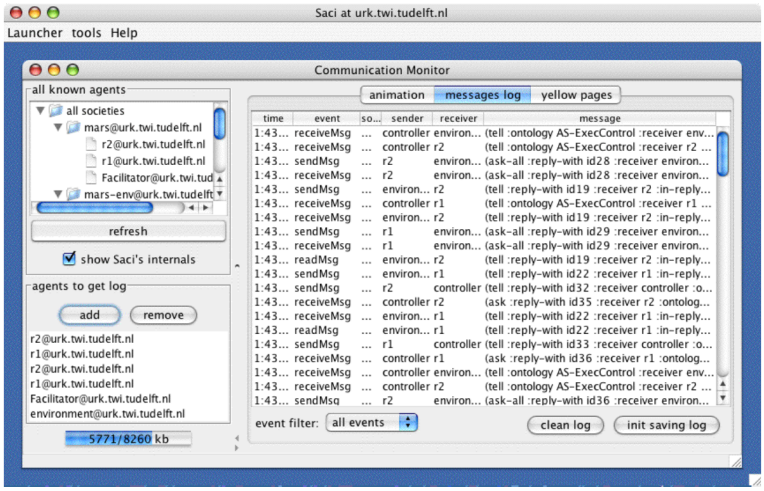

Figure 6. Saci message tracer.

2. Agent Frameworks

This report will inspect the following frameworks:

2APL platform [2apl] provides a set of tools that are designed to support the implementation, execution, and testing of multi-agent systems. In particular, the 2APL platform provides a graphical interface through which a user can load, edit, execute, and debug 2APL multi-agent programs using a syntax-colored editor, different execution modes, and several debugging/observation tools. The platform is built on top of the JADE platform allowing it to be run on several machines connected in a network.

JASON MAS (Java-based Interpreter for AgentSpeak) [Jason] provides support for developing Environments and support for MAS organisations and agents that reason about them, using the Moise+ model. It enables run a multi-agent system distributed over a network (using Saci or JADE); other distribution infrastructures can be added by the user. The IDE is in the form of a jEdit or Eclipse plugin ; the IDE includes a "mind inspector" which helps debugging.

JACK Development Environment [JACK] allows the definition of projects, aggregate agents and teams, and their component parts under these projects.

JADEX framework [JADEX] is based on the FIPA standard. The Jadex framework consists of an API, an execution model, and predefined reusable generic functionality. The API provides access to the Jadex concepts when programming plans. Plans are plain Java classes, extending a specific abstract class, which provides useful methods e.g. for sending messages. On top of JADE, The BDI Viewer tool allows to view the internal state of a Jadex agent, that is, its current beliefs, goals, and plans (see picture). The Jadex Introspector is similar to the JADE Introspector, allowing to monitor and influence the execution of an agent, by observing and influencing how incoming events are handled. For debugging purposes the Introspector also allows to put an agent into single-step mode. A Logger Agent is provided, which allows to collect and view log messages from JADE and Jadex agents, following the Java Logging API.

FLEEBLE4 Agent Framework [Fleeble] runs agents in a separate thread of the framework's process space. It has functionality to start, freeze, suspend, stop and kill agents.

2.1 2APL

I can not get 2APL to run completely. First, I can not run the current version of 2APL on OSX10.4 as there is no Java 1.6 available (you will only get an UnsupportedClass-VersionError exception). On Linux a similar problem, the system administrators have to install java 1.6 especially for this. 2APL does not run over my X server either, the only way to get this workable is to work at the console of the computer where Java6 has been installed. Even then, the system seems to be unstable.
I let the agents make a few steps, then I draw a simple world with some walls, bombs and trash cans, and I run the system. Drawing the world works without problems, but after starting the run the agents seem to hang after a dozen of steps. The system quickly continues to create over 10000 new states but the agents do not move at all. The problem seems to occur when one of the agents hits a wall, without walls it seems to work.
Agents in 2APL are JADE agents, and therefore most likely the scheduling behaviour is identical to plain JADE (see the JADE section above).
The 2APL Platform manager appears after start-up (Figure 7 etc). The platform manager provides means to (re)load an mas file, to step or run one or more agents, and to start JADE

tools like debugger, sniffer and RMA. Inside the jars I notice cartago software [Cartago], suggesting that 2APL can run on alternative platforms.
Below these platform function buttons are the information panel. A single agent can be selected with tabs on the left and various information about the selected agent can then be displayed with the tabs above the panel. The overview button shows contents of Belief- Goal and Plan base. Belief updates shows only the updates made to the belief base. PG- PC and PR-rules show the (prolog-like) rules of the agent. Selecting "Files" shows a list of used agent specification (.apl) files, with edit buttons for each. Pressing edit launches Jext, allowing you to edit the file. If you select the Mas tab you can edit the .mas file and check overall system messages. This seems to work correctly only when the platform is not running.

Debugging, Scheduling

Scheduling of the individual agents seems to be done by JADE. The 2APL platform has the option to halt threads, most likely it uses the JADE functionality to implement this. Resetting an agent seems not possible. New agents are probably launched and initialized using JADE.
You can independently step each agent. The states are maintained separately for each agent, so one agent may make much more steps than the other agent.
Stepping goes at the level of steps of the action-perception cycle. In the FGDC menu you can follow these steps, as shown in Figure 7. You can not step at deeper levels.
Resetting the entire MAS can be done with the "reload agent(s)" button or menu item. It seems that upon reload, the old agents are killed and new agents are injected into JADE. This also means that these agents will turn stale in opened JADE debugger- and sniffer windows, and that you have to manually reconfigure these windows.
Interestingly, the action-perception ('deliberation') cycle per agent can be modified directly, by adding rules and connecting them in the FGDC window. This is shown in Figure 8.

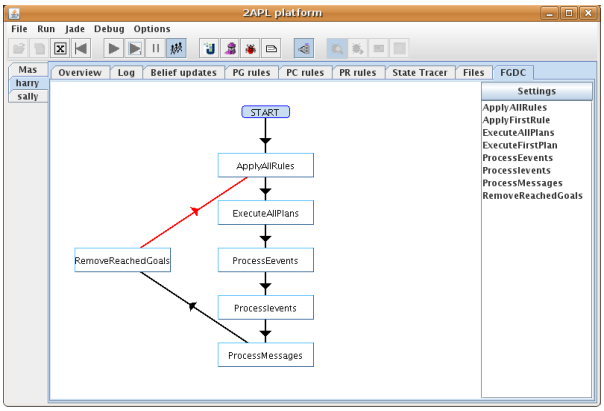

Figure 7. The red arrow indicates the next step that will happen if you press the one-step button (6th button from the left).

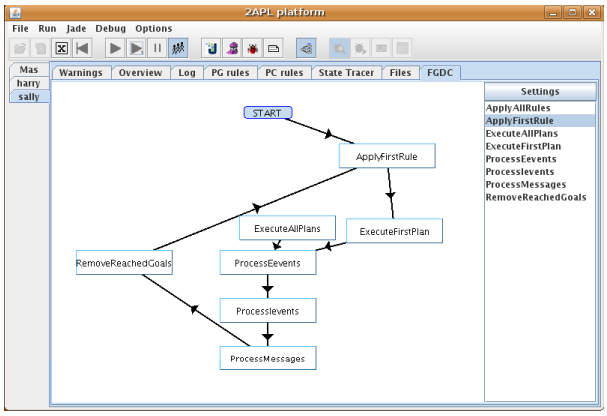

Figure 8. The action-perception cycle of each agent can be modified with the FGDC window.

The JADE introspector does not show these details of the action-perception cycle. It blocks on "DeliberationBehaviour" all the time (see Figure 9). Communication between the platform manager and the agents does work invisible to the JADE middleware platform. This makes me wonder whether JADE is completely ignored for this part and whether they 2APL is using its own message passing mechanisms beside JADE. It does keep the sniffer window nicely uncluttered with "system" messages, but that can also be achieved easily by not adding system agents to the list being sniffed. From the above, it seems that 2APL is only weakly linked with JADE.
Communication between agents works via the 2APL command

```
send(Receiver,Performative, Language, Ontology, Content).
```

We see a tight binding with the FIPA view of agent message content. Communication is a special action, distinguished from actions on the environment that always start with '@'.

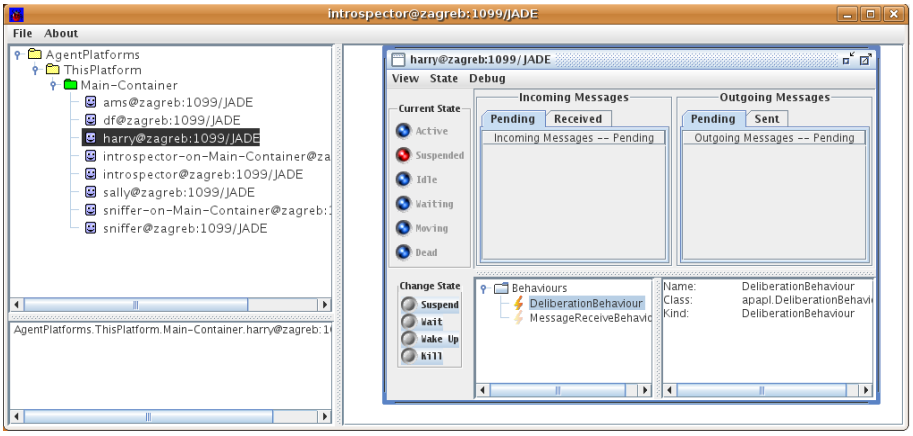

Figure 9. JADE Introspector viewing a 2APL agent.

The state tracer (Figure 10) looks overwhelming and messy with a large matrix of beliefs, goals, plans and logs. Every cycle, all columns are shifted left. Only a single agent can be inspected at a time. It looks better when you select less information with the checkboxes at the bottom.

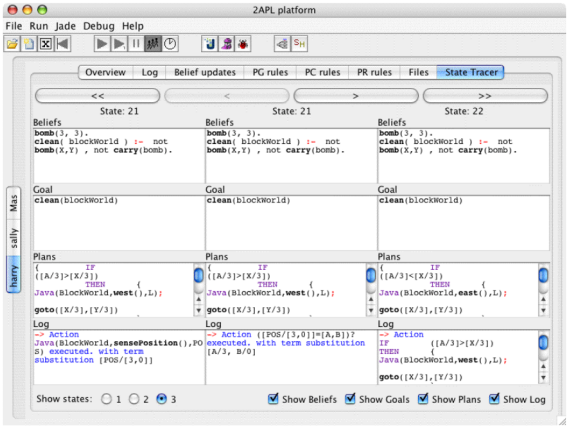

Figure 10. 2APL state tracer.

Modularization

Modularization is hard to determine, as we have no access to the source code. The core system is communicating with the user via the "2APL Platform" GUI (see Figure 7, Figure 8). The core program is not part of the JADE system.

The environment is a separate set of class files that can be loaded into the core system. The environment is not a JADE agent. The environments are found by name and have to be in a pre-defined directory. The environment must implement the functions addAgent and removeAgent to handle insertion/deletion of agents, and throwEvent that is used to send events (which are 2APLFunction objects) to the agents. If the 2APL program encounters a statement like @Env(<actionname>,Return,TimeOut) the java function <actionname> is called, and Return is bound to the return value of the Java function. Next,the 2APL call has to be converted and to a Java call and then the call has to reach the possibly remote computer where the environment lives. It is not clear whether and how this has been done. Maybe remote environments are not even supported. Calls to the environment are invisible to the JADE sniffer, so they are not using the JADE messaging system. So far I could not plug in remote containers into the 2APL system, maybe because I had not Java 6 on the remote machine. The most natural choice seems to use RMI but may not be actually used since Java

does not support time-out on RMI since version JDK1.2 (see http://www.jguru.com/faq/view.jsp?EID=332524).

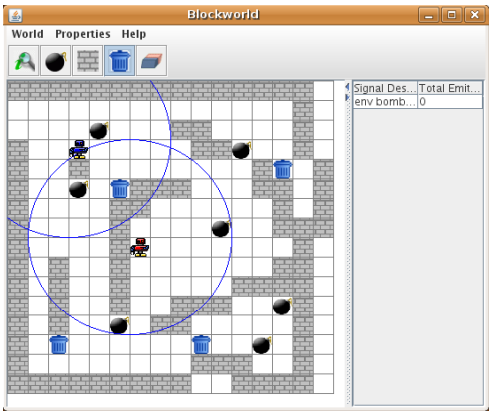

Figure 11. Blocks world environment provided with the 2APL examples.

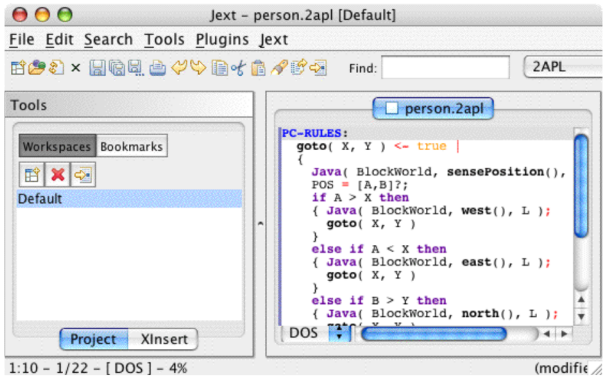

Figure 12. Jext editor window

For syntax directed editing, the Jext editor is being used (Figure 12). Basically this is a plain-text editor with parsing and keyword highlighting that can edit the .mas and the .2apl files used to build a set of agents and join them in a MAS.

GUI Design

Start-up works intuitively. Double click 2apl.jar to start the application. Sselect 'open' from the file menu and then select the .mas file from the example directory. All agents are loaded automatically and the world is ready to run.
Overall, a neat and mostly intuitive user interface. Most important platform functionality is available via the platform GUI. Agent mobility and multiplatform support seems not supported: there are no buttons or menu items for it. I can still manipulate the JADE platform into a multi-JVM configuration by using an external process (eg, issue a command to add a container straight to JADE, using a unix console) but the entire system becomes unstable if I add just a single container, and agents can not be moved to the new container.
Selecting an agent goes with tabs at the left border, while some of the buttons at the top of the window work only on the selected agent. This is not fully respecting the common interface guidelines. However it still works pretty intuitively and saves some space as they now share a line with other more globally working buttons.
Syntax highlighting is nice can be turned on globally. I just wonder why it is not just always on.
Only a single agent can be inspected at any time (Figure 7). This may make it harder to debug multi-agent systems where the agents are really cooperating.
The graphical designer interface for agent deliberation steps looks nice but I there seem just so few possible configurations that I wonder about its actual usefulness.
The possibility to see only the updates on the belief base is a nice idea. Especially if the belief base is a large, mostly static program and there are only a few updates to it.

2.2 Jason MAS

In Jason, agents are specified in agentSpeak. Sending messages is done with the dedicated 'send' predicate which is a special 'internal action'. There are many more internal actions like desire, intend, creating agents etc. Technically, agents communicate via JADE messages (FIPA messages).

Modularity

The JASON framework core is a parser and interpreter for agentSpeak, a beliefbase with set of actions, and a bit of execution control. This framework looks highly modular, with the core JASON functionality completely separated from both middleware (the agent platform that handles message passing, scheduling, etc) and editors. The beliefbase is not a separate module though. The global architecture picture for Jason looks like Figure 13.

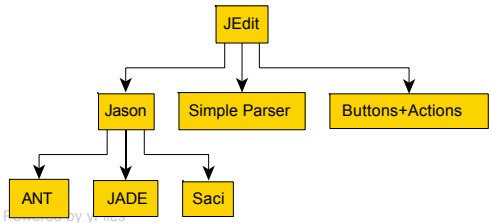

Figure 13. Global architecture of Jason.

The central controller at the top of the framework controls the loading of the agents and MAS, editing and running it. This part is integrated with the editor. This merging of editor and platform control functionality is a bit weird but seems common practice. This part is also modular, one can choose for either the jEdit editor or the Eclipse editor.
The MAS set-up is controlled with a .mas file. With the jEdit controller the .mas file can be edited straight away. With the Eclipse controller there is no visible .mas file, the MAS seems to start all agents available in the src/asl directory.

The JEdit central controller

The first version of the central controller is the plug-in for the jEdit file editor (Figure 14). The plug-in specifies a row of buttons to control Jason, and the Java functions to call when pressed. These buttons appear in the lower right half of the window. The plug-in also contains a simple parser specifying how to editor will display the text.

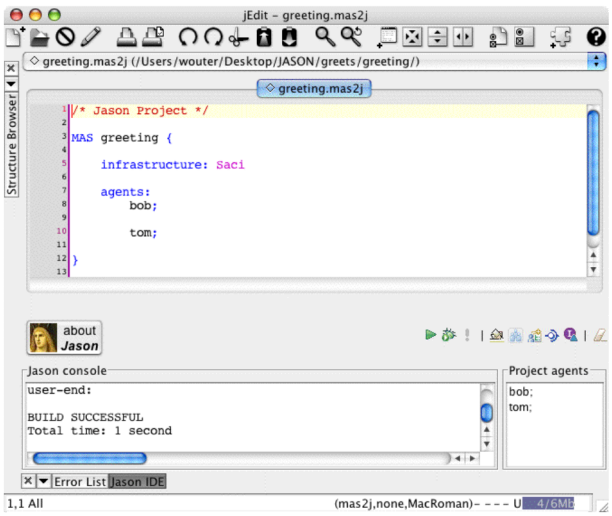

Figure 14. The jEdit file editor as Central Controller.

Jason has a number of configuration options to be set up properly (Figure 15). This is the way the modules are linked into Jason. Internally, the AgArchInfraTier and RuntimeServicesInfraTier serve as a middleware abstraction layer (Figure 16).

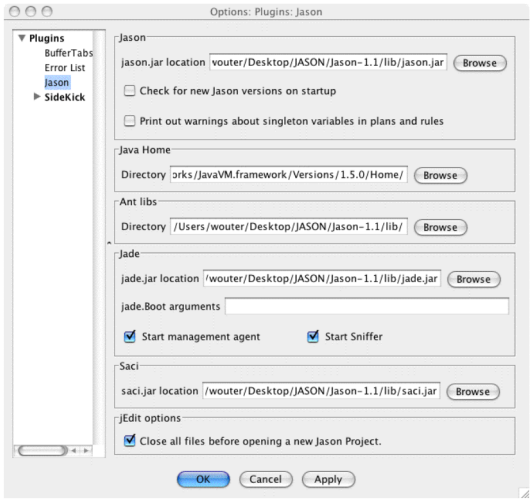

Figure 15. Jason configuration window for jEdit.

```
interface AgArchInfraTier {
 List<Literal> perceive();
 void checkMail();
 void act(ActionExec action, List<ActionExec>
       feedback);
 boolean canSleep();
 String getAgName();
 void sendMsg(Message m)
 void broadcast(Message m)
 boolean isRunning();
 void stopAg();
 void sleep();
 void wake();
 RuntimeServicesInfraTier getRuntimeServices();
}

interface RuntimeServicesInfraTier {
 boolean createAgent(...);
 AgArch clone(...);
 boolean killAgent(.. agName);
 Set<String> getAgentsName();
 void stopMAS();
}
```

Figure 16. Jason Middleware and environment abstraction layer.

Eclipse central controller

The second version of the central controller is via the plug-in for Eclipse (Figure 17). The eclipse platform creates a Java and an asl directory in the src directory. All agents are created in the asl directory. So far I could not find the configuration window (Figure 15) when running under Eclipse. Therefore JADE control windows do not open and interaction with the platform is severely hampered (eg no JADE debugging).
Eclipse has a sophisticated plug-in architecture [eclipseplug02]. It may provide good ideas for further modularization of the GOAL system as well.

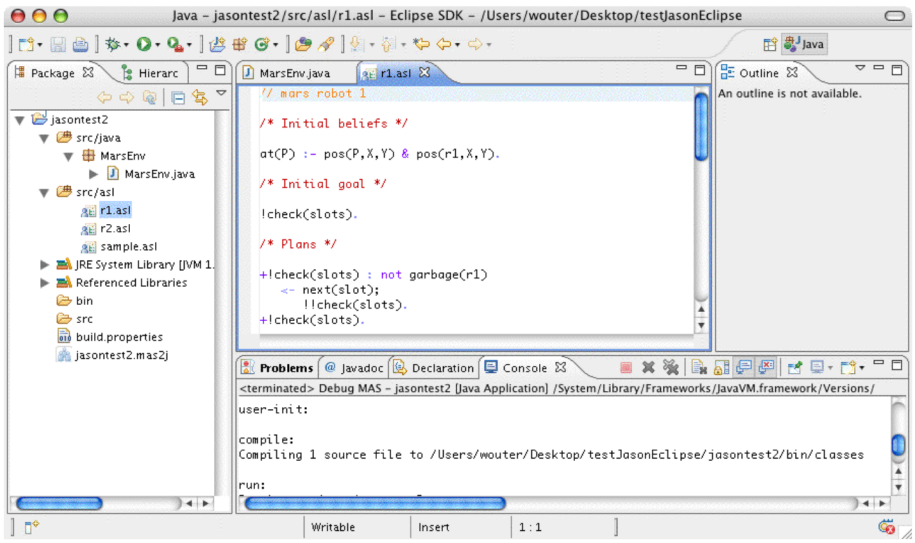

Figure 17. Eclipse as Jason Central Controller

Run-time controllers

As soon as you click on a run or debug button in the editor, the JASON framework launches additional processes to handle the runtime environment, and a number of GUIs to control them. These GUIs fall in three categories: the agent inspectors, the environment and infrastructure controllers.
Scheduling of the agents seems under control of JADE, with JASON using calls to JADE to (attempt to) impose its own scheduling (like stepping one agent). Agents can be created from JASON but also from JADE level.

Agent Inspectors

The Jason agent inspectors consist of the MAS Console that shows basic feedback text (Figure 18), and the JASON Mind inspector shows the status of the agents (Figure 19). The mind inspector appears only when running in debug mode. The actual layout and contents in the MAS Console depend on the selected infrastructure.

The JASON Mind Inspector allows inspection of past states, and stepping per agent or for the entire platform. The Mind Inspector appears only when selecting 'debug' from the Central Controller. This debug construction is sometimes conflicting with the debug facilities offered by the infrastructure (see below).

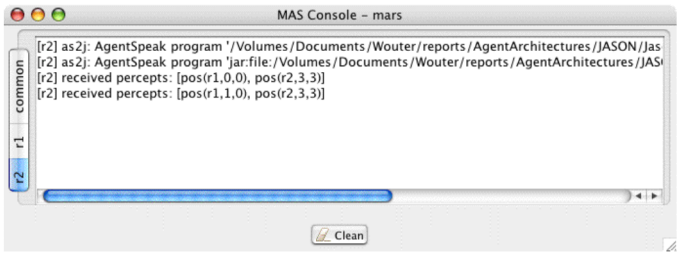

Figure 18. The JASON MAS Console

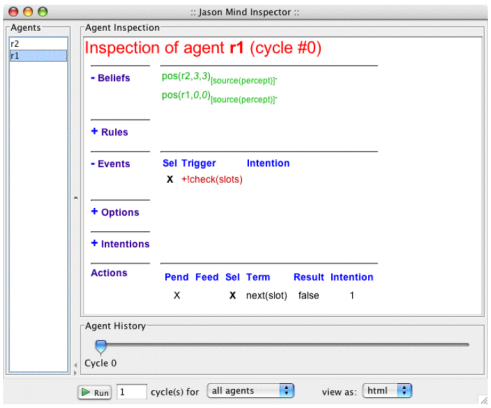

Figure 19. The JASON Mind Inspector.

Environment

The environment can launch any GUI it wants. Figure 20 shows an example, from the Mars environment where two robots cooperate on cleaning the world. The environment is a just another JADE agent. This means that requests for actions can be sniffed, debugged (stepping etc) and introspected as well. The Mars environment does not have a very informative behaviour though (just one CyclicBehaviour).

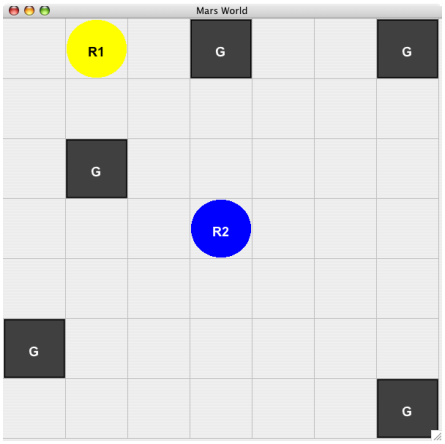

Figure 20. The Mars environment from the Jason examples.

Infrastructure control

Additional platform control windows pop up depending on the type of platform and options that were chosen. JASON can currently run on three infrastructures: JADE, Saci and a Centralised infrastructure. This selection is made in the infrastructure selector in the mas2j file. Saci and JADE are independent platforms, the Centralised infrastructure seems to have been developed by the JASON programmers.

If you select JADE, the entire range of platform controls, such as the sniffer (Figure 3), debugger and platform controller (Figure 4) become available.

Debugging, Scheduling

Debugging processes seem to depend lightly on the chosen system for the infrastructure (Jade, Saci, Centralised). Resetting an agent or a platform from within the runtine environment is not possible. So resetting has to be done by quitting the runtime and restarting it.

JADE

If the JADE platform is chosen, the Jason core seems to work together with the JADE platform via a j_controller agent inside the JADE platform. The usual debugger (Figure 1), sniffer (Figure 3) and platform controller (Figure 4) can be used.
In this case, debugging controls appear also in the Console window, as in Figure 21. Debugging can get highly confusing with this system. If you halt an agent with the JADE debugging facilities and then ask JASON to do a step, the Run button goes dead. If you then resume the JADE agent, neither the agent nor the Run button does not come alive anymore.

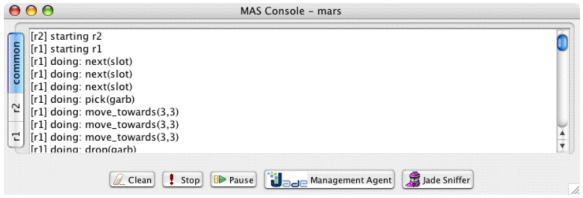

Figure 21. Console window with JADE.

Messages to the environment are plain text and well readable (Figure 22). Messages between agent and the j_controller are sometimes unreadable (Figure 23), but there are also readable messages such as requests to perform a cycle.

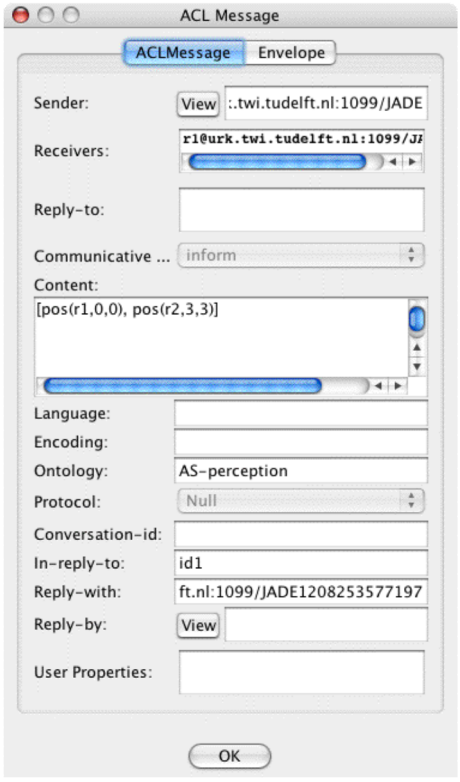

Figure 22. JADE message from environment to agent, responding to a

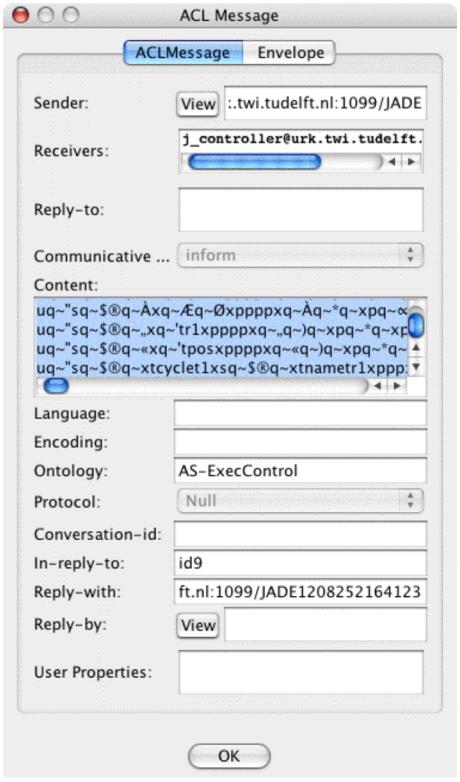

Figure 23. Unreadable response from an agent on an agState query.

getPercepts query. **agent on an agState query.**

Saci

When using SACI there appear no platform controller buttons in the console, as in Figure 18. Most other functionality seems identical.

Centralised

With the "centralised" infrastructure, the basic platform control buttons are added to the MAS Console (Figure 24) as with JADE. As the Mind Inspector window also has a run button, debug controls are getting a bit confusing. Pressing the Pause button in the MAS console appears to pause only one agent, not the entire system.

Figure 24. Platform control buttons added to the MAS Console.

GUI discussion

A drawback of the high modularization of Jason is that the various GUIs do not form a seamless whole. The JEdit and Eclipse editor windows have a very different look and work very differently than the configuration window or Jason Mind Inspector and the JADE windows.
Another apparent[3] drawback of this is that separate runtime windows are launched when the application is started. This also has an advantage, as the same windows can be launched when running the application separate from the editor. We discuss this further in the discussion at the end of this document.
Both editors have a row of vague icons at the top. As already discussed with the JADE RMI interface (Figure 4), this is mostly useless to me.

2.3 JACK

I use a Linux system for testing. I tried to run remotely, but there seems a problem with X11. Therefore I tested from the console (Ubuntu 7.10 on Linux 2.6.22-14 with Gnome 2.20.1).
The Hanoi example compiles without problems but you need to follow the manual carefully, as the provided compile and run scripts are needed. Lots of files are created, for instance you start with 26 files in the hanoi example, which blows up to 105 files after building). The Hanoi example by default runs with only an environment window (Figure 25), without any signs of a way to communicate with the platform manager (if there is any?).
The flow example does not seem to work. A lot of windows open at start, but when I press the 'continue' button in the time control window the current time increases just one second, then all windows become unresponsive and nothing happens anymore. When I set the update rate to 0.1 (instead of the default 0.001) I get a few seconds running but the application crashes again.

[3] Apparent because it is not clear whether the editors are actually preventing further integration or the programmers of Jason did not know how to do this.

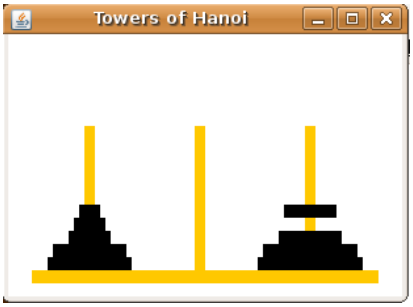

Figure 25. The Hanoi example running.

The hanoi example can not be opened in the development environment, as there is no project file. The countingteam example can not either. The manual mentions the possibility to view the interactions with a -D option, but the application is finished within a second, just long enough to see a window open and close again. So far for intuitivity.
The manual indicates that a project file can be generated with a oneliner on the console:

```
java aos.main.JackBuild -r -x -E myproject.prj -dj backup
```

However trying this with the countingagent results in error messages about syntax errors in the .plan files. I tried on windows as well, with the Hanoi example this oneliner works but I get the same error messages with the counting example.
At this point it is determined that the Linux version of JACK is not stable. The windows version is downloaded and installed on Windows XP Professional 2002 SP2. On Windows the oneliner works without errors and a project file appears that can be opened in the JACK IDE (Figure 26).Running the application is not trivial. You have to know which .class file contains the main program, open the Compiler utility, go to the "Run Application" tab, select that class file, and then you can press the "Run" button in that tab.

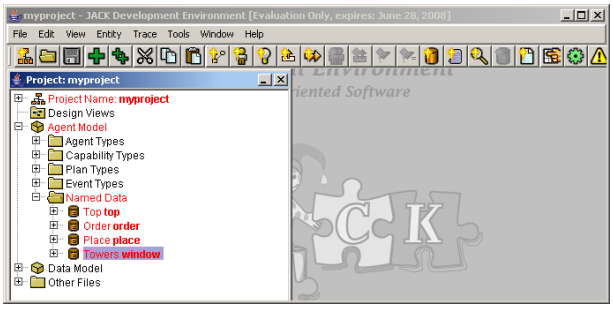

Figure 26. The JACK IDE window.

GUI comments

The interface is pretty much fixed size, and if you downscale the main window buttons will just disappear and windows inside the main window will just disapear behind the borders of the main window (not even scroll bars available, scroll wheel on mouse does not work either). The large trees run out of the bottom of the window and become unreachable, no scrollbar here either (Figure 27).
You need the right mouse button to add for instance event types, or to change the name of an object selected from the project list (Figure 27, left subwindow). Windows to edit these objects sometimes appear as a subwindow, and sometimes as an external window.
Again there is a large row of vague icons at the top. There is no 'run' button available.

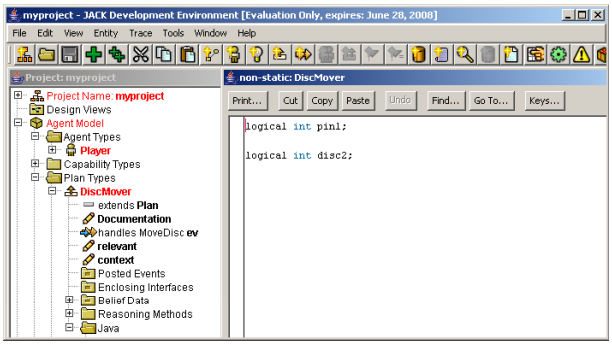

Figure 27. Windows and trees run out of the main window. This causes buttons to disappear and tree parts to become unreachable.

On the left is a huge tree, aparently trying to show the entire agent structure in an unfoldable tree. It looks overwhelming and I don't see much added value over a good structured text file. It is not clear how the system behaves with multiple agents, as the one-liner to create project files failed for most examples.

Debugging

If you want to do tracing, you must select "Trace Graphical Plans and Events" before pressing the "Run" button (Figure 28 left). After you press the run button the tracing application windows appear (Figure 28 right). You can now follow the steps being taken, the active parts turn blue in the graph and additional info can be selected via the check boxes. Unfortunately the tracing program crashes on the Hanoi example (all tracing windows lock up) after a few steps after I select all check boxes. It seems not possible to debug at deeper levels, debugging is mainly looking at plan steps. There is a combo box "Tracing" but you can only trace 2 events.

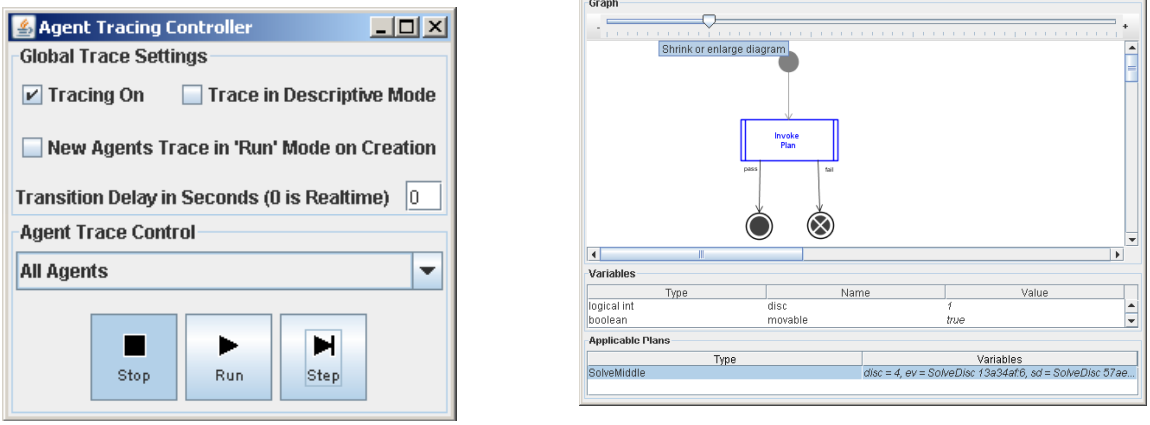

Figure 28. After selecting "Trace Graphical Plans and Events" two windows appear to enable tracing. Left the step/run interface, right the plan step tracer.

The manual mentions that it is possible to trace/debug without having the IDE. I assume that this is done with the right command from the console and that the windows of Figure 28 then appear.
There seems no way to reset agents or environment from the runtime environment. The only way to reset is to completely restart the simulation.
Jacksim does not work either. The mkit and runit commands are not even .bat files, causing errors when trying to run them. After fixing this, it appears that the java files in jsv/table have

not been compiled. Compiling them does not help in the end, a file jsv.table.Table.java does not exist and it is needed somewhere.
JACK provides a syntax directed editor for Java files as well as a graphical plan editor. The syntax directed editor is based on the Jacob project manager for Emacs [Jacob02]. All agent specification files appear to be Java style files anyway with one extra keyword, a dash (#). There is a whole set of special instructions to support inter-agent communication.

2.4 JADEX

The central access point for JADEX is the Jadex Control Center (Figure 29). It provides access to standard tools: introspector, tracer (ala JADE sniffer), message center (JADE DummyAgent). Looks like there is JADE under water, I see the typical JADE names "df", "ams". Nevertheless the manual mentions that running on JADE is optional and needs the JADE adapter.

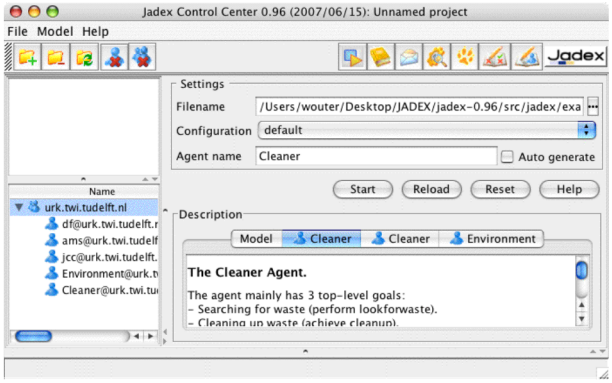

Figure 29. JADEX Control Center

Modularity

I did not deeply investigate this but it looks like the JADEX platform is built as a set of JADE agents. To set up and run a MAS, you manually have to load an environment "agent" and worker-agents into the system. Then you start the environment agent, followed by the other agents.

Environment

I used the cleanerworld as test environment (Figure 30). Nice environment, and the agent moves very smooth through this environment. Also funny is that agent, when not active anymore, slowly fades out (first his yellow blob, then the agent himself). Plugging in agents and environment goes intuitively.

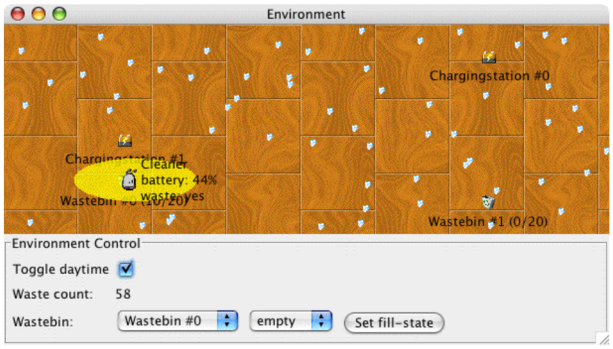

Figure 30. The cleanerworld environment and cleaner agent.

Debugging, Scheduling

Jadex is controlling which agent is running and which one stepping. Each agent can be stepped, halted or run independently. Debug steps reach much deeper than just the message level, many internal actions can be stepped on (Figure 31). This is much more elaborate than the other platforms so far. The detail steps only involve ProcessEventActions thate are decomposed in substeps FindApplicableCandidatesAction etc. (see Jadex Tool Guide p.22). Agent messages can also be traced. This works by placing a traceradapter around an agent. The source code of Jadex is available but large and it takes too much time to check this in detail.

The Control Center shows Reload and Reset buttons. However pressing "Reset" does not reset the selected agent but only removes all selected agent files. In fact the runtime keeps running after pressing reset. The Reload button seems not to do anything at all. Pressing "start" again launches another agent of the type selected.

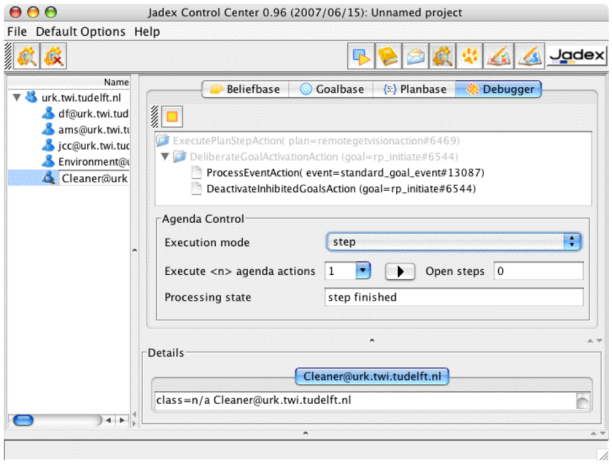

Figure 31. Jadex introspector, selected the debugger and set step mode for Cleaner.

Belief base changes can be watched in real time (Figure 32). The view on the plan base seems a bit limited (Figure 33), you get lots of statistics but you can not view the actual Java code behind the plan.

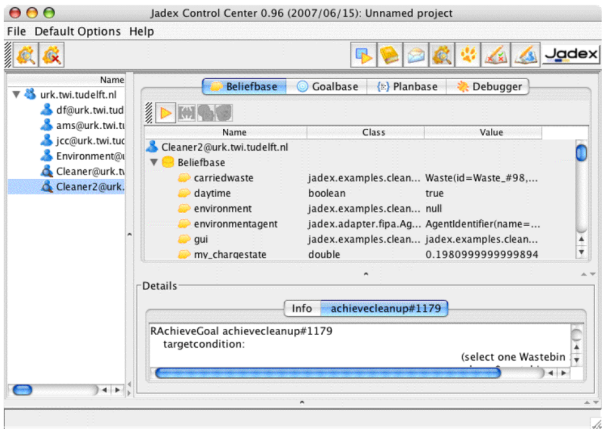

Figure 32. Belief base contents, showing realtime changing data.

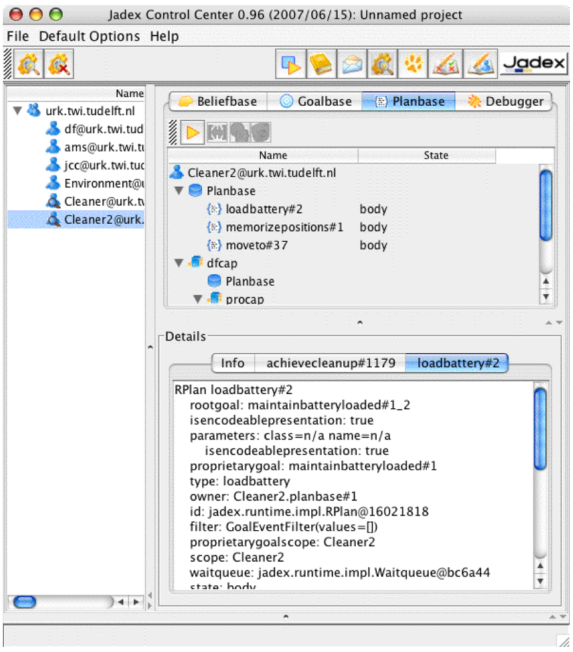

Figure 33. Planbase view. No Java code can be inspected.

The entire system is event based (Figure 34), starting with events that change beliefs, goals, etc. I could not find hard statements about the execution model, but agents seem to run independently, no signs of unusual scheduling mechanisms. If running on Jade the agents must be independent threads, and with the default (non-jade) configuration the agents also seem independent threads in a JVM. If two agents are run in parallel, the second agent is fading in and out all the time in the environment, and the CPU load goes to 100% for the Java thread. Apparently these agents are very CPU intensive.

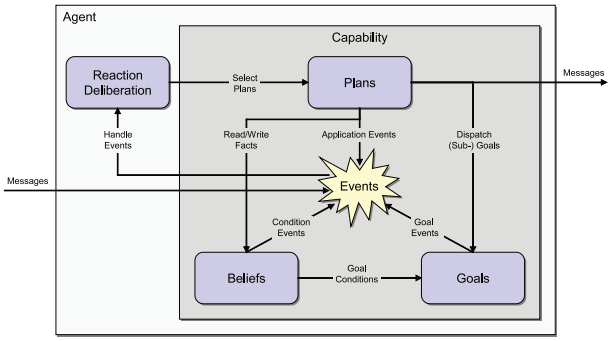

Figure 34. JADEX Architecture is event based.

Communication with other agents runs directly via sendMessage calls from Java. The environment is also an agent. Strangely, the environment (Environment.java) is just a static object inside the JVM. Each agent just gets its copy of the static object and calls the functions inside the environment object to perform actions. The environment apparently reports the events explicitly to the control center. This seems just not correct in multiprocessor environments, but I did not test that.
You can edit the belief base at runtime. There seems no support to edit the agent definition files or the Java files defining the plans. There appear not to be MAS files.
There is a large number of components and it is not clear how they are interwoven. Everything appears to be included straight as source .java files, with hardly any jars.

2.5 FLEEBLE

The fleeble.net website reports that the OSX version is broken. Therefore I work with the Linux version. The Linux download does not want to load agents (testing on Ubuntu 7.10 on Linux 2.6.22-14 with Gnome 2.20.1 and Java 1.6). Adding tools.jar to the CLASSPATH does not help. Finally the tools.jar issue is resolved by correcting the FLEEBLE preferences and UNchecking the "Find tools.jar on startup". Next, we run into a java.lang.UnsatisfiedLinkError: no swt-pi-gtk-3128 in java.library.path. This appears an old issue with Eclipse-generated files (https://bugs.eclipse.org/bugs/show_bug.cgi?id=88669). First we try to install the right 3128 in the java.library.path explicitly but this fails. Then we find out that there is a new version libswt-pi-gtk-3236 for this machine. As a quick hack we rename that to pi-gtk-3128 and then this problem is finally fixed.
Next we run into a problem with the serialization and Java1.5. Dmytro had fixed this and sends the new version. But this new version does not want to load: some java class version ids are rejected as out of range. Dmytro also remembers that there is a java bug with the serialization in Java1.6.
The Windows version installs and starts without problems. Double clicking the Fleeble icon starts the Fleeble Client (Figure 35). The client is just to control the run-time core, and has no editing capabilities. However it works with the .java files (instead of with class files) and recompiles them every time an agent is started or reloaded. Although nice, this obviously also poses security issues (everyone can read your source code). Also I wonder whether

dependencies (other java files being included) and special compile options are handled correctly.

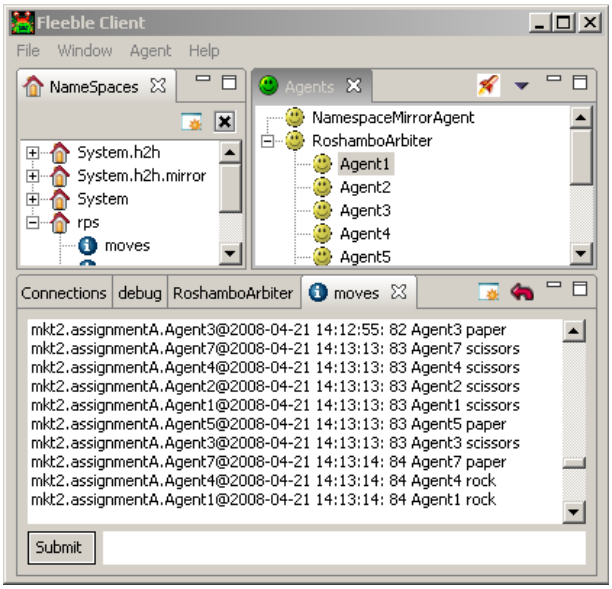

Figure 35. Fleeble Client GUI.

I received a number examples from the mkt2 project from Dmytro. For example, using the File menu of the client to load Roshambo Arbiter from assignment 1 opens a tournament runner (Figure 36).

Figure 36. The Roshambo Arbiter GUI.

Architecture

The core is the 'fleebleserver'. It implements compilation of the implemented agents. The core also implements a notion of 'containers', similar to JADE, that handles message handling and agent manipulation (move, create, delete, etc). Agents speak with the containers via an Ambassador object.

The Fleeble Client GUI (Figure 35) is a separate interface connecting to it. According to the manual you can also run without the server. The client runs in a separate JVM to keep it responsive when a fleeble agent running on the server starts eating all CPU cycles (of the JVM on which the server runs).

Agents are separate threads in (potentially multiple) JVMs. Agents are loaded by the Ambassador, and they initialize themselves. The autonomy of agents is restricted to the process space of its parent agent (or the platform). Every message delivered on a message channel spawns a new thread for every agent subscribed to that channel [Pantic05].

Communication can be done only over defined channels, using send and receive. If I get it right, when subscribed to X, the Java callback function handleX of agent A is called whenever a message of type X arrives at A. The environment is in the examples just another agent, usually the parent agent, and standard messages are used to communicate with that environment.
I think customized 'scheduling' is possible, by having agents communicate exclusively with the Arbitration agent that organizes the 'game', and having that Arbiter sending requests and waiting for responses according to the planned schedule. The core does not provide alternative scheduling.
There seems no support for .mas files. For example the Roshambo application is hard coded to load agent 1 to 9.

Tracing, Debugging

In the Linux versions we encountered a lot of problems. It was a bit confusing to have to search through the namespace tree in order to found the error messages.
Tracing messages is possible by right-clicking the agent to track, and then select an input or output channel. Strange is that you get messages from ALL agents, even if you place only two in the competition (). Maybe I don't get the interface.
You can also right-click an agent and select 'freeze', 'suspend' and 'run'. However, if you freeze an agent the tournament player seems to crash, nothing happens if you play, also not if you run the just-frozen agent and even not when you press play again.
An agent can also be Reloaded. [Pantic04] indicates that this recompiles the source and replaces the running agent(s) with the new version. I could not test this as I have no working demo where agents live long enough to do a reload.

3. Overview

This section gives a very brief overview of a number of key differences between the various agent platforms.

Table 2 shows the variation in the reset functionality on the various platforms. The middleware we investigated does really support resetting of agents, in spite of some functions in their interface that seem designed for this. On many platforms, a reset involves killing and restarting the entire middleware layer, and reloading the entire MAS, or even quitting the entire system. Only Fleeble supports reloading of individual agents (according to the manual, we could not test it). On 2APL a reset reloads the entire MAS, making it unclear whether the middleware was restarted or not.

Table 2. Reset functionality of the agent platforms.

	JADE	Saci	2APL	JASON	JACK	JADEX	FLEEB
Reset of **A**gent, **M**AS possible			M	M			A
Reset: Middleware keeps **R**unning or **Q**uits			R?	Q			R

Table 3 shows the main similarities and dissimilarities in the GUI structure of the various agent platforms. Most platforms stay close to the JADE model and provide similar tools. In this model, messages are the way the various agents communicate. Often the environment is another agent in this model. In many systems the overall GUI structure is messy, because various modules all have their own GUIs and widely different look and feel.

Table 3. Similarities and dissimilarities in the GUI structure. Join style can be separate windows (SEP), one window with tabs (TAB), or windows-in-window (WIW). empty fields indicate not available or not applicable.

	JADE	Saci	2APL	JASON	JACK	JADEX	FLEEB
IDE available to edit agents			Y	Y	Y		
Join style of IDE and RT Core GUIs			Joined	Sep.	Sep.		
Tools: **R**MA, **S**niffer, **I**ntrospector, **E**vent tracer	RSI	RS	RSI	RSI	E	RSI	
Join style of Tools into single GUI	SEP, WIW	WIW	TAB[4], SEP	SEP, TAB	TAB, SEP, WIW	TAB	TAB
Step/Run buttons in which GUI	intro spector		main win.	intro spector	trace contr.	agt. panel	
using "right mouse button" (not available on OSX)	Y	Y	N	N[5]	Y	N	N
error / warning message output per **A**gent or **G**lobal or per **C**hannel	G	G?	A	G	?	A	C (?)

[4] JADE tools sometimes appear, and these use WIW

[5] Eclipse, one of the super-components, and sub-component JADE do use right mouse buttons.

Table 4 shows the external software to support the GUI construction used by various platforms. Referred modules probably use more external modules, for instance Eclipse in turn probably uses SWT. We did not check this.

Table 4. External GUI-related modules used by the platforms.

Platform	External modules used for GUI
JADE	Swing
Saci	
2APL	[jExt],
JASON	[JADE], [Saci], [jEdit], [Eclipse]
JACK	[BlueJ], [jEdit]
JADEX	[BCel], [TouchGraph]
Fleeble	[SWT]

References

[2apl] http://www.cs.uu.nl/2apl.

[apple07] Apple Human Interface Guidelines (2007). http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/XHIGIntro/chapter_1_section_1.html

[BCel] The Apache Jakarta Project: BCel. http://jakarta.apache.org/bcel.

[BlueJ] BlueJ – The interactive Java environment. http://www.bluej.org.

[Cartago] Cartago. http://alice.unibo.it/cartago

[Eclipse] Eclipse - an open development platform. http://www.eclipse.org/

[Fleeble] Fleeble Agent Framework. http://www.fleeble.net

[Gallardo02] Gallardo, D. (2002). Developing Eclipse plug-ins: How to create, debug, and install your plug-in. Available Internet: http://www.ibm.com/developerworks/opensource/library/os-ecplug.

[JADE] Java Agent DEvelopment Framework. http://jade.tilab.com.

[Jacob02] Jacob – the Java Commando Base. Available Internet: www.kclee.de/clemens/jacob.

[JACK] Jack. http://www.agent-software.com/products/jack/index.html.

[JADEX] Jadex BDI Agent System. http://vsis-www.informatik.uni-hamburg.de/projects/jadex.

[Jason] Jason: a Java-based Interpreter for an extended version of AgentSpeak. http://jason.sourceforge.net/

[jEdit] jEdit Programmer's Text editor. http://www.jedit.org

[jExt] Jext Free source code editor. http://www.jext.org/

[Pantic04] Pantic, M., Grootjans, R. J., & Zwitserloot, R. (2004). Fleeble Agent Framework For Teaching An Introductory Course In Ai. IADIS International Conference Cognition and Exploratory Learning in Digital Age (CELDA 2004).

[Saci] http://www.lti.pcs.usp.br/saci.

[SWT] SWT: The Standard Widget Toolkit. http://www.eclipse.org/swt/.

[TouchGraph] TouchGraph: Data Visualization Made Practical. Old version available http://sourceforge.net/projects/touchgraph. New version probably through http://www.touchgraph.com.

[UXG] Windows Vista User Experience Guidelines. http://msdn2.microsoft.com/en-us/library/aa511258.aspx