Making Software Agent Technology available to Enterprise Applications

Dick Cowan, Martin Griss
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2002-211
July 26th , 2002*

E-mail: dick_cowan@hp.com, martin_griss@hp.com

applications, application server, agents, J2EE, JADE

Software agents provide the technology necessary to dynamically negotiate, select, and utilize the appropriate services required in today's highly dynamic business world. In this paper, we describe our research into combining an agent platform with a J2EE application server to enable the use of software agents in enterprise applications. This provides the necessary first step of integrating agent technology with other mainstream web server, application server, DB, communication and security technologies, to produce a scalable, robust platform for intelligent applications of various kinds.

* Internal Accession Date Only
Approved for External Publication
© Copyright Hewlett-Packard Company 2002

Making Software Agent Technology available to Enterprise Applications

Dick Cowan
Hewlett-Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304 USA
01-650 857-8038
dick_cowan@hp.com

Martin Griss
Hewlett-Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304 USA
01-650 857-8715
martin_griss@hp.com

ABSTRACT
Software agents provide the technology necessary to dynamically negotiate, select, and utilize the appropriate services required in today's highly dynamic business world. In this paper, we describe our research into combining an agent platform with a J2EE application server to enable the use of software agents in enterprise applications. This provides the necessary first step of integrating agent technology with other mainstream web server, application server, DB, communication and security technologies, to produce a scalable, robust platform for intelligent applications of various kinds.

Categories and Subject Descriptors
D.4 [**Operating Systems**] – Application Servers
I.2.11[**Distributed Artificial Intelligence**]: multiagent systems

General Terms
Management, Performance, Design, Reliability, Experimentation, Standardization.

Keywords
Applications, application server, J2EE, JADE

1. INTRODUCTION
We will provide an overview of our work to make the JADE [1] agent platform become a managed service under a J2EE [2] application server. We used HP's application server (HPAS version 8.0)[3] for the following reasons:

- As stated on the HP-AS web site [3]:

  HP-AS was built, "from the ground up," on a completely standards-based, modular architecture and provides developers with the unprecedented ability to "pick-and-choose" only the specific services they want or need. This allows a software developer to leverage the knowledge and effort of experienced developers and/or avoid recreating components to common software design requirements. HP-AS provides the core set of services that service-based applications require, including naming and directory, management, logging, and security services.

- It is freely downloadable [3], making it easily accessible to the agent development community.

We use the term ***BlueJADE*** to refer to the combination of HP-AS (formerly known as Bluestone Application Server) with the JADE agent platform.

Although FIPA [4] standards addressed agent to agent communication and agent hosting environments, no agent platform implementation to date has reached the quality of commercial software applications, and in particular one of the key obstacles to the widespread deployment of agent technology is the relative immaturity of agents in regard to scalability, federation, persistence, transactions, security, deployment lifecycle, management, and integration with legacy systems or existing systems. These features are crucial to provide robust, reliable agent-based intelligent applications. Current systems leave the agent developer with a lot of work to do when building a real application, and this often is not done well, if at all. Such ***industrialization*** work is essential for any deployed application.

2. PLATFORM INDUSTRIALIZATION
Agent platforms have traditionally looked at themselves as the complete support system in which agents do their work. Although this works for simple agents, developers are left without a portable solution when it came to interacting with traditional web services or appliances. As shown in Figure 1, an agent platform should correctly be viewed as simply one of many services offered to the enterprise application developer.

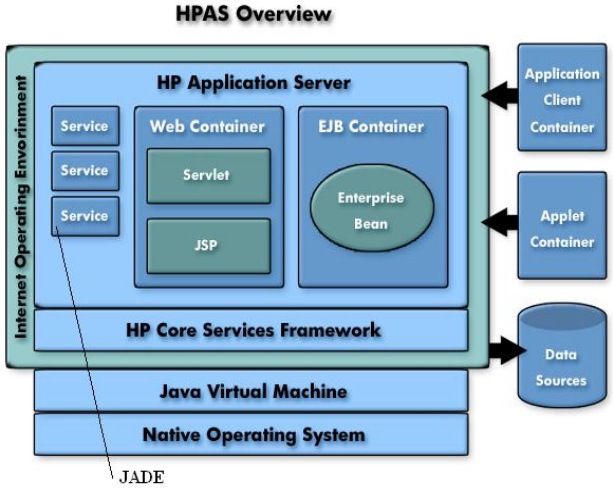

Figure 1. Showing the JADE service within HP-AS

By "management" we include a number of typical system, application and service configuration, monitoring and control actions, such as: a) detecting and restarting a faulty element; b) detecting load conditions and adjusting resources or moving an element to another processor; c) restarting an element when its configuration changes; d) collecting information on the normal and abnormal client and resource state, statistics and usage of each component; and e) element lifecycle monitoring and control. By "element" here we mean a single agent, a related group of agents, an agent container[1], a complete agent platform, or even a set of agent platforms on multiple machines with associated other software.

In order to do this, we must provide a "standard" API that such management systems can access. As an example, one might also consider SNMP, J2EE management beans [5] or the DMTF WBEM and CIM [6].[2] A companion project is using an SNMP compatible Agent MIB to allow HP OpenView to manage JADE agents and platforms [15].

3. Specifications of the Agent Hosting Service

To develop an agent hosting service that can be managed there are two primary components that we consider:

3.1 Agent facing

Although we are currently using JADE, if possible we wanted the work done on this side to be as applicable as possible to other (FIPA-compliant) agent implementations. As such, we wanted to limit our view of the underlying agent system to just those interfaces exposed by a package we could influence. For the JADE effort this package is named **jade.wrapper**. We envision that as agent systems increase in popularity that some of the interfaces defined in this package would move into a standards body (such as FIPA or JAS) and become standard interfaces that agent platform builders would implement and on which agent application builders could rely. They would then become part of some neutrally named package. We also recognized that our requested changes would need to done in a manner that permitted the agent platform to function as a stand-alone entity with no dependencies on any particular server or framework.

3.2 Server facing

This component is the one seen by the software that desires to manage and host the agent platform. To enhance the applicability of this component to different agent platform implementations, the classes in this component should restrict their view of the underlying agent system to just those interfaces and/or classes defined in the first component.

4. Agent Facing Implementation

For the JADE implementation, we added our classes to the package ***jade.wrapper*** as it already had classes there to control an agent as well as their containers. Our work started in the summer of 2001 using JADE version 2.2. Working closely with the JADE developers we contributed our work, in open source form, to them. These enhancements are now part of version 2.5. In this section we describe the classes we modified or added to this package.

An agent management service must provide the ability to manage at least two entities: the agent ***platform*** and individual ***agents***. It is desirable to implement platform management in a manner that (optionally) hides individual implementations, such as JADE's containers or agent groups.

The interfaces **PlatformController** and **AgentController** define methods used to control the platform or individual agents. These interfaces are implementation neutral. The exception **ControllerException** is thrown by methods of both interfaces. The class **PlatformEvent** is used to notify platform listeners of platform events.

4.1 State Related Classes

Both the agent platform and individual agents have life cycles and hence specific states. Although JADE's **core.Agent** class defines life cycle states, to adhere to implementation neutrality we introduced state classes to the wrapper package and then let the JADE specific implementation of our agent controller map between our generic states and theirs.

4.2 Agent Management Services

The interface **AgentController** defines the following methods for agent management:

- getName – Get agent name.
- start – Start agent.
- suspend – Suspend agent.
- activate – Resume following suspend.
- kill – Terminate the agent.
- getState – Get the agent's state.

The exception **ControllerException** may be thrown by the above methods. Platform specific exceptions (like JADE's **StaleProxyException**) would be caught and re-thrown as a **ControllerException** by the class implementing **AgentController**. This is very similar to **JasException** proposed by JAS [7].

The class **Agent** in the JAPE's **wrapper** package serves as a management proxy to an actual agent and provides a JADE specific implementation of **AgentController**.

4.3 Platform Management Services

The interface **PlatformController** defines the following methods for platform management:

- getName – Get platform name.
- start – Start platform.
- suspend – Suspend platform.
- resume – Resume following suspend.
- kill – Terminate the platform.
- getState- Get the platform's state.

[1] By Agent container here, we mean some sort of agent or element grouping capability, such as the JADE container or J2EE container [2].

[2] At this point, it appears that the JAS [7] Version 1.12 specification does not address management in this broader sense, but may have some interfaces that might influence a future iteration of our design.

- addPlatformListener – Add an event listener.
- removePlatformListener – Remove a listener.

Platform listeners implement the following methods, all of which have a **PlatformEvent** argument containing details of the specific event.

- bornAgent – An agent was born.
- deadAgent – An agent died.
- startedPlatform – The platform was started.
- suspendedPlatform – The platform was suspended.
- resumedPlatform – The platform was resumed.
- killedPlatform – The platform was killed.

The class **AgentContainer** in JADE's **wrapper** package provides a JADE specific implementation of **PlatformController**.

5. Server Facing Implementation

To create a service to run under HP-AS as well as package everything in a manner suitable for external distribution, we have created a stand-alone tree containing documentation, all source core, and two sample applications.

5.1 Service Implementation

The service implementation consists of the following components:

1. JADEService.java – Every service requires an interface and implementation. This provides the interface for the service.
2. JADEServiceImpl.java – The implementation of the service. This class uses the agent facing code described earlier.
3. Logger.java – Capture System.out and System.err output and place it into HP-AS's log with date, time, thread name, and message type header.
4. JADEServiceImpl.mbean – Defines those methods that may be managed through any JMX [8] browser.
5. JADEServiceImpl.properties – Provides properties about this service.

To activate the service requires the addition of a description of the JADE service to be added to HP-AS's XML deployment definition file. This description assigns the service a name, provides the fully qualified class name of the class that implements this service, and specifies the name of its configuration file. It is in this latter XML configuration file that one specifies arguments to the service, arguments to JADE, and defines the collection of agents to be started along with their arguments.

Using this XML configuration file has provided a simple and extensible solution for defining properties of individual agents. The first boolean valued property we defined is used to indicate if the agent should be automatically restarted should it abnormally terminate. The second is to indicate if the agent requires a GUI and to not start it if no display console is available (as would be the case when starting HP-AS without a GUI). Another attribute we have discussed is one to control how many instances of a particular agent to start. Clearly there are other interesting attributes that can be defined and used to control agents and the platform.

5.2 Demonstration Applications

As part of the distributed package we have included two simple applications that demonstrate agent control and communication between JSPs and agents.

5.2.1 DemoAgent

Contains two simple agents which exchange messages. The sender agent sends a QUERY-REF message to the receiver agent every 10 seconds. The receiver agent responds with an INFORM message with the same message content as it received. This repeats indefinitely. JADE's RMA agent is also started from which their Sniffer can be run to view the individual message exchanges. This application also makes available two JSPs that may be invoked from any browser. The first JSP will send a message to the receiver agent (via a socket bridge agent) and display the response message. The second JSP creates another sender agent instance that also sends to the existing receiver.

5.2.2 Agentcities

Starts a single Ping agent that will communicate with other agents over an HTTP transport protocol. This application provides the functionality necessary to launch a new Agentcities [10] site. In addition the URL http://localhost:9090/agentcities will cause HP-AS to present a simple form that allows one to send an Agentcities information request to an Agentcities node and display the response.

6. Other Platforms

Clearly there are other agent platforms as well as J2EE application servers and what was we have done could be redone with other combinations. We are currently looking into a port to the open source J2EE server JBoss [11]. It is our hope that agent and agent platform management will be addressed by FIPA such that minimal lifecycle management would be defined in a consistent manner. Attempting to add management to an existing implementation presents many interesting challenges. Considering the different agent-hosting environments (J2ME/CLDC to J2EE) [12] and using the terminology of the FIPA Abstract Architecture Specification [13], one would say, "an agent **may** be manageable". This could be done by having an agent return its management interface (which could be null) or to always create agents with a platform factory method which would return agents with appropriate management functionality for their hosting platform.

For Java based implementations JMX [8] would be the obvious standard to follow. By initially thinking of the fundamental components as managed beans (MBeans), the complete system becomes very easily managed.

7. Status and Next Steps

We have an initial working implementation of BlueJADE running on a collection of machines under Windows NT, Windows 2K, and Linux. Using the standard HP-AS control mechanisms we can start and stop the JADE agent platform as well as individual agents. We can also demonstrate automatic agent restart and platform reconfiguration – whereby if you modify the XML file defining the active agent population the changes will impact what is actually running in the agent platform.

At the time of this writing one can manage a JADE main container and agents in it. The JADE extensions to enable management of an agent container (vs. a main container) are still

pending. In typical usage with HP-AS one would usually create and manage a JADE main container anyway. We are now using this work to support four in-house sandbox systems on which we run the assortment of services and agents used daily by researchers in our group [9]. It is also supporting our four Agentcities sites [14] with others under construction. We have packaged BlueJADE in a form permitting simple distribution and installation by other interested parties and are currently working with a few selected organizations[3].

We encourage wider usage, and contributions from the agent community in this important area of agent industrialization. Here are a few open questions for future work:

- Agent restart likely needs some form of "exponential backoff" whereby should the agent fail on a subsequent restart the time to its next attempted restart is increased.
- What is the best way to load balance agents? Can agents be forcibly moved to benefit load balancing?
- How should agent persistence be handled so as to integrate well with data base services?
- What is the relationship between an agent and a JSP?
- When an agent needs to act on your behalf and interact with legacy systems requiring password access how should that be done so as not to compromise security?

8. ACKNOWLEDGMENTS

We thank David Bell, an original member of the team, for his effort in starting this work. We would also like to thank Kevin Smathers (HP Labs) for all his help with our Linux testing environment and Michael Li (HP Corp Infrastructure) for the BlueJADE trail map, and proposing some of our next step activities. We also appreciate the efforts by members of our Bluestone middleware division for answering questions concerning HP-AS and its components. Finally, we would like to acknowledge Fabio Bellifemine (Telecom Italia Lab) and other members of the JADE team for working with us to help shape and implement the modifications to JADE.

9. REFERENCES

[1] Fabio Bellifemine, Agostino Poggi and Giovanni Rimassi, "JADE: A FIPA-Compliant agent framework", Proc. Practical Applications of Intelligent Agents and Multi-Agents, April 1999, pg 97-108 (*See http://sharon.cselt.it/projects/jade for latest information*)

[2] J2EE information and specifications. http://java.sun.com/j2ee/

[3] HP-AS information and download. http://www.bluestone.com/products/hp-as/default.htm

[4] FIPA: Foundation for Intelligent Physical Agents, see http://www.fipa.org , and P.D. O'Brien and R. Nicol, "FIPA: Towards a standard for intelligent agents." BT Technical Journal, 16(3), 1998.

[5] J2EE Management Beans (JMX) http://java.sun.com/products/JavaManagement/wp/

[6] WBEM/CIM. See http://jcp.org/jsr/detail/48.jsp

[7] Java Agent Services Specification, version 1.0. http://www.java-agent.org/Documents/documents.html

[8] Java Management Extensions http://java.sun.com/products/JavaManagement/

[9] Agents for Mobility department: http://www.hpl.hp.com/org/stl/maas/index.html

[10] Agentcities: http://www.agentcities.net/

[11] JBoss J2EE server. See http://www.jboss.org/

[12] J2ME/CLDC See http://java.sun.com/products/cldc/

[13] FIPA Abstract Architecture Specification http://www.fipa.org/specs/fipa00001/

[14] Salt Lake City, UT. See: http://agentcities.cs.utah.edu/agentcities/ Palo Alto, CA. See: http://mml.hpl.hp.com/agentcities/palo-alto/ Honolulu, HI. See: http://ewasx.hpl.external.hp.com/agentcities/ Miami, FL. See: http://huphu.hpl.external.hp.com/agentcities/

[15] Brian Remick and Robert Kessler, "Managing Agent Platforms with AgentSNMP," CS dept, U of Utah, April 2002. Submitted to this workshop.

[3] This work is being done by our research group [9] to better understand the many important issues relating to agent usage by J2EE applications and should not be interpreted as any future product plans or commitments on the part of HP.