BlueJADE – A service for managing software agents

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

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

Bernard Burg
Hewlett-Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304 US
(650) 857-2403
bernard_burg@hp.com

Abstract

In this paper, we describe our research into transforming an agent platform into a manageable service that can be managed by a modern application server. This is a key part of industrializing agent technology, that is, 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.

Keywords: Software agents, management, scalability, industrialization, application server, JADE.

1 Introduction

This document provides the motivation and overview of the work done to enable HP's Bluestone Application Server (HP-AS) [1] to manage an agent service using the Java Agent DEvelopment Framework (JADE) [2]. Internal to HP, and in this document, we use the term ***BlueJADE*** to reflect this technology combination. BlueJADE aims to provide an industrial grade agent environment, which can be managed as any other application in an application server, and comply to both the FIPA Abstract architecture and the FIPA 2000 specifications. We believe that BlueJADE is a major milestone towards the success of agent technology deployment.

2 From Proprietary Agent Technology to a Widespread Global Deployment

The key property of agents is to communicate with others, whoever and wherever they are. This communication problem was overlooked in the past as initial agent technology was entrenched in incompatible proprietary technology, where most of the agents were unable to interoperate and were therefore defeating their very purpose of existence. Today the only viable solution is to deploy agents on a global scale.

Agents needed a standard allowing them to communicate and interoperate. The FIPA [8] standard was created in 1997 and brings this much-needed interoperation model to agents. The next step was to provide freely accessible implementations of standard agent environments to develop a critical mass around the technology, and prepare a market for future deployment. Seven open source implementations operating in various environments running from servers to small appliances; as well as a Java Community Process on Agent Services (JAS) [7] of the FIPA standard [8] are available.

However, none of these implementations has yet 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.

The most important step to the success of agent technology is a large-scale deployment of an open, freely accessible testbed to share the technology, gather feedback and generate the critical mass of users. This effort is underway, and Agentcities is currently building a worldwide network of agent platforms communicating and working together on a 24/7 basis [3].

The world of agents is seeking an industrial grade agent environment, and we believe that BlueJADE is a major milestone. To the best of our knowledge there is no competitor to BlueJADE today.

3 HP's Contributions in Agents and Application Servers

3.1 Agent Technology

The Agents for Mobility Department within HP labs [6] has been experimenting with agent technology using the JADE agent system. Its effort is focused on personal assistants (agents which act on behalf of their owner) using a profile of the user to

accomplish specific tasks. Our first demonstration was to schedule a meeting using the participant's preferences and calendars as well as room availability. This seemingly simple scenario actually provides a rich environment for different threads of investigation:

- It involves a surprisingly large collection of agents and services. We are already using components of HP-AS to provide JSP support for meeting setup requests or to handle email notifications to invitees.
- It provides multiple opportunities for intelligent failure processing. For example when an invitee's personal assistant is unreachable.
- It demands intelligent selection and usage of those services used by ones personal assistant. For example ones personal calendar may in fact reside in two differing implementations: work related using Microsoft Exchange vs. personal using Palm Desktop.
- It would benefit from data persistence and transaction restart or roll back.
- It requires a concerted effort to support an appropriate security model for allowing agents to access applications and information on behalf of their users.

The applications developed in the Agent for Mobility Department target mobile users, and therefore these agent-applications need to run on small appliances as well as on servers. This motivated the choice of JADE over other agent platforms; since JADE leaped to small appliances with LEAP, the second generation platform capable to run the core of JADE on J2ME CLDC MIDP profile. LEAP is already running on phones and has plans to migrate towards watches. It is also worth mentioning that both JADE and LEAP are distributed under the classic LGPL open source license and communicate seamlessly.

With this background, this report puts its focus on the J2EE application server world for its considerable stability and because its adherence to standards has existed for some time. In addition, this environment allows us to grow the functionality of agents towards the semantic web, UDDI and SOAP, so as to lead towards an integrated environment combining the best of each technology. For this opportunity to become reality, agent platform services (so called "agent management") must be viewed as a service that may be managed along with all other services by the application server.

It is worth noting that our approach moves away from a purely agent-centric vision towards a more heterogeneous agent integrated vision, where agents become the proactive accessor of the future semantic web, leading it from a static set of knowledge towards a proactive entity serving the user.

In the agent-centric vision, one wanted to solve all problems through agents, and this lead to duplicate and redundant efforts. In the more modern vision of the agent-integrated vision, agents are using existing solutions; typically for manageability of applications. To achieve this, the best choice is to integrate agent platforms into application servers allowing them to benefit from existing experience and deployment.

3.2 Web Server Technology

The rapidly growing widespread popularity of the efforts to industrialize a Java-based application server and web-server technology, using J2EE/EJB technology is under way. HP's version of this is called HP Application Server that may be freely downloaded [1]. By leveraging this technology we solve many of the industrialization issues, provide a base for solving other issues, and most importantly begin to properly partition responsibilities of effort. Clearly, from the enterprise view, agent technology should be viewed as just another manageable service. Making JADE into a service manageable by HP-AS is just the first necessary step to be capable of leveraging all the other services.

4 Our Agent-Management Objectives

By "management" we include a number of typical system, application and service configuration, monitoring and control actions, such as:

- detecting and restarting a faulty element
- detecting load conditions and adjusting resources or moving an element to another processor
- restarting an element when its configuration changes
- collecting information on the normal and abnormal client and resource state, statistics and usage of each component
- 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 [4] or the DMTF WBEM and CIM [5].[2]

The benefits of making the JADE agent platform operate well as a HP-AS service include:

- More robust intelligent web-service platform, for e.g., Agentcities
- Additional resources to help industrialize the technology
- Distribution and advocacy, leading to a 'de facto' default standard platform for research and deployment
- Make link to other standard activities like W3C, DAML+ OIL, UDDI
- Make BlueJADE become a strong proposal of the Java Agent Services
- Make a beachhead in the world of agent-skeptic people

5 Specifications of the Agent Hosting Service

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

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 recognize that our requested changes must be done in a manner that permits the agent platform to function as a stand-alone entity with no dependencies on any particular server or framework.
2. **Server facing**. This component is the one seen by the software that desires to manage 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.

6 Agent Facing

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 in their 2.4+ snapshot builds and will be included in the 2.5 release. In this section we describe the classes we have 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.

[1] By Agent container here, we mean some sort of agent or element grouping capability, such as the JADE container or J2EE container [4].

[2] At this point, it appears that the JAS 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.

6.1 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.

There are four classes relating to life cycle state as follows:

1. State - Interface defining those methods that apply to all state objects.
2. StateBase - Abstract base class providing common state related functionality.
3. AgentState - State object for agents.
4. PlatformState – State object for platforms.

6.2 Agent Management Services

To manage individual agents one uses the following:

1. AgentController – Interface defining the following methods for agent management:

```
public String getName() throws ControllerException;
public void start() throws ControllerException;
public void suspend() throws ControllerException;
public void activate() throws ControllerException;
public void kill() throws ControllerException;
public State getState() throws ControllerException;
```

2. ControllerException – Thrown by methods defined in the AgentController interface. 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].

Agent – This class in the wrapper package serves as a management proxy to an actual agent. It provides a JADE specific implementation of AgentController.

6.3 Platform Management Services

To manage the agent platform one uses the following:

1. PlatformController – Interface defining the following methods for platform management:

```
public static interface Listener[3] extends EventListener {
   public void bornAgent(PlatformEvent anEvent);
   public void deadAgent(PlatformEvent anEvent);
   public void startedPlatform(PlatformEvent anEvent);
   public void suspendedPlatform(PlatformEvent anEvent);
   public void resumedPlatform(PlatformEvent anEvent);
   public void killedPlatform(PlatformEvent anEvent);
}

public String getName();
public void start() throws ControllerException;
public void suspend() throws ControllerException;
public void resume() throws ControllerException;
public void kill() throws ControllerException;
public AgentController createNewAgent(String nickname, String className,
                                      Object[] args) throws ControllerException;
public State getState();
public void addPlatformListener(Listener aListener) throws
                                      ControllerException;
```

[3] This interface defines a second interface to be used when adding or removing a platform listener. This was done to facilitate implementation neutrality. This interface would be extended to cover other platform events such as agent movement.

```
public void removePlatformListener(Listener aListener) throws
                                                    ControllerException;
```

2. ControllerException – Thrown by methods defined in the PlatformController interface. Platform specific exceptions would be caught and re-thrown as a ControllerException. This is very similar to JasException proposed by JAS [7].
3. AgentContainer – JADE specific implementation of PlatformController.

7 Server Facing

Our implementation uses HP's Application Server [1] version 8.0. To create a service to run under HP-AS we created a package named ***com.hp.bluejade***[4] that contained the following:

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. JADEServiceImpl.mbean – Defines those methods that may be managed.
4. JADEServiceImpl.properties – Provides properties about this service.

To activate the service required that the following be added to HP-AS's deployment definition file:

```
<Service name="jade" class="com.hp.bluejade.JADEServiceImpl">
  <Registers>
    <Interface name="com.hp.bluejade.JADEService"/>
  </Registers>
  <Provides>
    <Package name="com.hp.bluejade"/>
  </Provides>
  <Configuration url="jade-service-config.xml" reload="30"/>
</Service>
```

This service descriptor assigns it a name (jade), provides the fully qualified class name of the class that implements this service (com.hp.bluejade.JADEServiceImpl), and specifies the name of its configuration file (jade-service-config.xml). The attribute reload="30" causes HP-AS to monitor the file every 30 seconds for changes. If changes are made, our service will be notified and will examine the new collection of agents and take appropriate actions. As shown below this file describes JADE's initialization arguments as well as describing the 3 agents (name, class name, and attributes) that should be started.

```
<?xml version="1.0"?>
<!-- JADE must be started with nomtp true so it won't try to
    create a CORBA ORB service which is already provided by HPAS. -->
<jade-service
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation=
        "http://hplstl1.hpl.hp.com/xml/schemas/JADE-1_0.xsd"
        args="-dump container:false nomtp:true"
        note="All agents except RMA will auto restart.">

        <agent name="echo" class="com.hp.agent.testing.Echo"
                restart="true" />

        <agent name="PingAgent" class="com.hp.agent.ping.PingAgent"
              args="import:${CONFIG_LOC}/coolagent.properties"
                restart="true" />
```

[4] The name of this package isn't important. Classes in this package only rely on JADE and HP-AS jars. Our plan is to contribute these classes and the few configuration files to JADE for inclusion in their jade/add-ons/ directory.

```
        <!-- The following for testing only -->
        <agent name="RMA" class="jade.tools.rma.rma"/>

    </jade-service>
```

As shown, for each agent you may specify its setup arguments as well as specifying if the agent should be automatically restarted should it abnormally terminate. The string ${CONFIG_LOC} will be replaced by its corresponding environment variable value using a simple filter program we created.

For our testing system, we actually start many more agents.

8 Status and Next Steps

We have an initial working implementation of BlueJADE capable of running our meeting arranger demonstration. 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 (wherein the collection of started agents is changed).

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. We are now using this work to support or in-house sandbox systems on which we run the assortment of services and agents used daily by researchers in our group. It is also supporting our two Agentcities [3] sites [9] with others under construction.

When available as an early JADE 2.5+ snapshot build we anticipate wider usage, and contributions from the agent community in this important area of agent industrialization. Here are just four examples of future work:

1. What is the best way to load balance agents? Can agents be forcibly moved to benefit load balancing?
2. How should agent persistence be handled so as to integrate well with data base services?
3. What is the relationship between an agent and a JSP?
4. 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?

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 trailmap, 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.

REFERENCES

1. HP Application Server (HP-AS) version 8.0. May be freely downloaded from http://www.bluestone.com/
2. 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*)
3. Agentcities: http://www.agentcities.org/
4. J2EE Management Beans (JMX) http://java.sun.com/products/JavaManagement/wp/
5. WBEM/CIM. See http://jcp.org/jsr/detail/48.jsp
6. Agents for Mobility department: http://www.hpl.hp.com/org/stl/maas/index.html
7. Java Agent Services Specification, version 1.0. See http://www.java-agent.org/Documents/documents.html
8. 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.

9. Salt Lake City, UT. See: http://agentcity.cs.utah.edu/web/slc/index.html
Palo Alto, CA: See: http://mml.hpl.hp.com:8080/