Experiences in the management of an EJB-based e-commerce application

Juan I. Asensio[†], Víctor A. Villagrá, Jorge E. López·de·Vergara, Roney Pignaton, Julio J. Berrocal.

Department of Telematic Systems Engineering
Technical University of Madrid (DIT-UPM). Spain.
{jasensio,villagra,jlopez,roney,berrocal}@dit.upm.es
[†] Visiting researcher from U. of Valladolid. Spain

Abstract

This paper describes the ongoing work within the MKBEEM (Multilingual Knowledge-based European Electronic Marketplace) European project devoted to the management of an EJB-based (Enterprise Java Bean) electronic brokerage service to be used in e-commerce environments. The paper focuses on the adopted management architecture, the tested application management instrumentation approaches and the management integration problems that appeared.

Keywords: EJB-based applications management, JMX, management instrumentation.

1 Introduction and Motivation

The evolution of distributed object-based applications technologies (CORBA, RMI, DCOM,...) towards the so-called distributed component-based development paradigm (related to "server-side" component models such as CCM[1], EJB[2], MTS[3], ... and their corresponding platform architectures: CORBA 3, J2EE, COM+...) has been recognised as a major step towards the achievement of the desirable goal of developing scaleable, distributed, secure, modular, transactional object-based applications, but allowing the developers to focus their attention only to the business-related aspects of those applications [5].

There is an increasing proliferation of platforms for the deployment of component-based server-side applications. Many of them are intended to be used as the basis for ASPs (Application Services

[1] CORBA Component Model [6]

[2] Enterprise Java Beans [8]

[3] Microsoft Transaction Server, http://www.microsoft.com/com/tech/MTS.asp

Providers) where reliability, performance and security are of great importance. Therefore, the management of component-based applications must be considered as a key point.

When dealing with component-based server-side applications, several management issues should be addressed:

- Management of the component-based platform
- Management of the application-independent functionalities (deployment, undeployment, persistence management, transaction management, etc.)
- Management of application-dependent aspects (i.e. management of service-oriented aspects of the applications).
- Management of the underlying system and network resources.

This paper describes some of the main management related aspects of an EJB-based application that is being developed within the scope of the MKBEEM (Multilingual Knowledge-Based European Marketplace[4]). MKBEEM is an European project that belongs to the IST (Information Society Technologies) program whose main goals consist of developing an intelligent, knowledge-based multilingual electronic commerce platform as well as validating and assessing prototypes of that platform on a pan-European scale. The ongoing development of the first prototype of the MKBEEM platform is based on the J2EE [11] platform that incorporates the EJB component model. More concretely, the reference implementation of the J2EE platform provided by Sun has been used, version 1.2.1, as the component platform for supporting the MKBEEM prototype.

Currently, the J2EE platform specification does not define how to solve the management problems commented above. Although several J2EE platform implementations include ad-hoc support for more or less advanced management capabilities, that is not the case of the Sun's J2EE reference implementation. As a result, the MKBEEM development team had to develop its own approach to the management of EJB applications. The first results from that work are the subject of this paper. More concretely, the paper describes the main characteristics of the adopted management architecture, which is based on the JMX (Java Management eXtensions) [9] management standard, and the approach to the management instrumentation of the components of the MKBEEM prototype, which could be introduced transparently to the developers of the functional aspects of the prototype.

As there are several J2EE platform vendors that have already proposed solutions related to the J2EE/EJB management problem, section 2 enumerates and briefly describes the most important ones. Section 3 explains the concrete work in the J2EE/EJB management problem within the MKBEEM project: section 3.1 details the adopted management architecture while section 3.2 discusses several instrumentation alternatives and points out the one selected for the MKBEEM case.

[4] http://mkbeem.elibel.tm.fr/

Finally, section 4 concludes the paper summarising its main ideas and proposing several ways of enhancing its results.

2 Management of EJB-based platforms and applications: existing approaches

The current specification of J2EE does not define management capabilities. There is an ongoing effort within the Java community (described in the corresponding "Java Specification Request" [2]) in order to specify and standardise all the application-independent aspects of the "J2EE management". Those aspects include the management of J2EE server resources as well as the management of applications lifecycle. The "Java Specification Request" for "J2EE management" requires that the proposed solutions are be based on standard management technology such as JMX, SNMP and CIM/WBEM. JMX seems the most promising approach due to its straightforward integration with Java (with respect to instrumentation) and the possibility of using protocol adapters and connectors for interoperability with other domains using different management technologies [9][4]. The first draft specifications of "J2EE management" is expected for October 2001.

Meanwhile most of the J2EE-compliant platform vendors include ad-hoc management solutions in their products. These solutions can be classified in two main groups:

- Manageability based on the addition of SNMP agents: Sun/Netscape's iPlanet Application Server 6.0, Borland AppServer 4.5 (through its Borland AppCenter 4.1), IBM WebSphere Application Server 4.0 (through its Tivoli Manager for WebSphere Application Server), etc.

- Manageability based on the addition of JMX-based agents: jBOSS (an "Open Source" project for implementing a J2EE platform), BEA WebLogic Server 6.0, IONA iPortal 1.3, etc.

  In this case, it is worth mentioning that the jBOSS platform uses JMX as the basis for its J2EE container. Each J2EE service is an MBean (Management Beans) and therefore they can be managed through the corresponding MBean Server from a Web Browser[5].

In both cases, manageability is limited to the J2EE services (naming, transaction, etc.) and to different stages of the applications lifecycle (deployment, packaging, etc.). In other words, it seems that there is not a unified way (if any) of accessing to management information related to service-oriented aspects of the application (i.e. application-dependent management information).

[5] http://www.jboss.org

The access to that kind of management information is usually related to the use of APIs for allowing the developers of the applications to expose the interesting management data. Nevertheless, there are other ways of instrumenting EJB-based applications that take advantage of some of the particularities of this kind of applications in order to hide to the developers most of the management related activities.

The following section describes how two aspects of the J2EE/EJB management problem have been faced in the management of the MKBEEM prototype:

- The design of the management architecture.

- The design of the management instrumentation of service-oriented management aspects within the managed application.

3 MANAGEMENT OF THE MKBEEM EJB-BASED APPLICATION

3.1 MANAGEMENT ARCHITECTURE

JMX was chosen as the basis for the management architecture (depicted in Figure 1) of the MKBEEM EJB-based prototype due to the reasons described in the section 2. Management information and operations of Java applications managed with JMX are made available through MBeans (management beans). According to the JMX specification, MBeans should be “plugged” to a MBean server that resides in the same Java Virtual Machine than the application.

Nevertheless, this scenario covered by the JMX specification was not possible in MKBEEM. The J2EE server and the JMX MBean server had to be run in different JVMs (they were two different products). Therefore, the management instrumentation of EJBs of the MKBEEM application in the form of MBeans seemed to be useless as they couldn’t be connected to the corresponding MBean server.

A double workaround was adopted to avoid this problem:

- For obtaining management information from the EJBs or for invoking management operations on them:

  One possibility was coding Standard MBeans that, upon the reception of an invocation of a particular management operation, would contact the remote EJB (by using a reference to its remote interface) in order to perform the requested operation. This approach would imply the codification of one particular “proxy” MBean for each application EJB.

A refinement of this approach consists of using the so-called JMX Model MBeans. These Model MBeans are Dynamic MBeans (MBeans that expose their management interface at run-time) that can be used to instrument Java code that does not follow the Standard MBean design patterns.

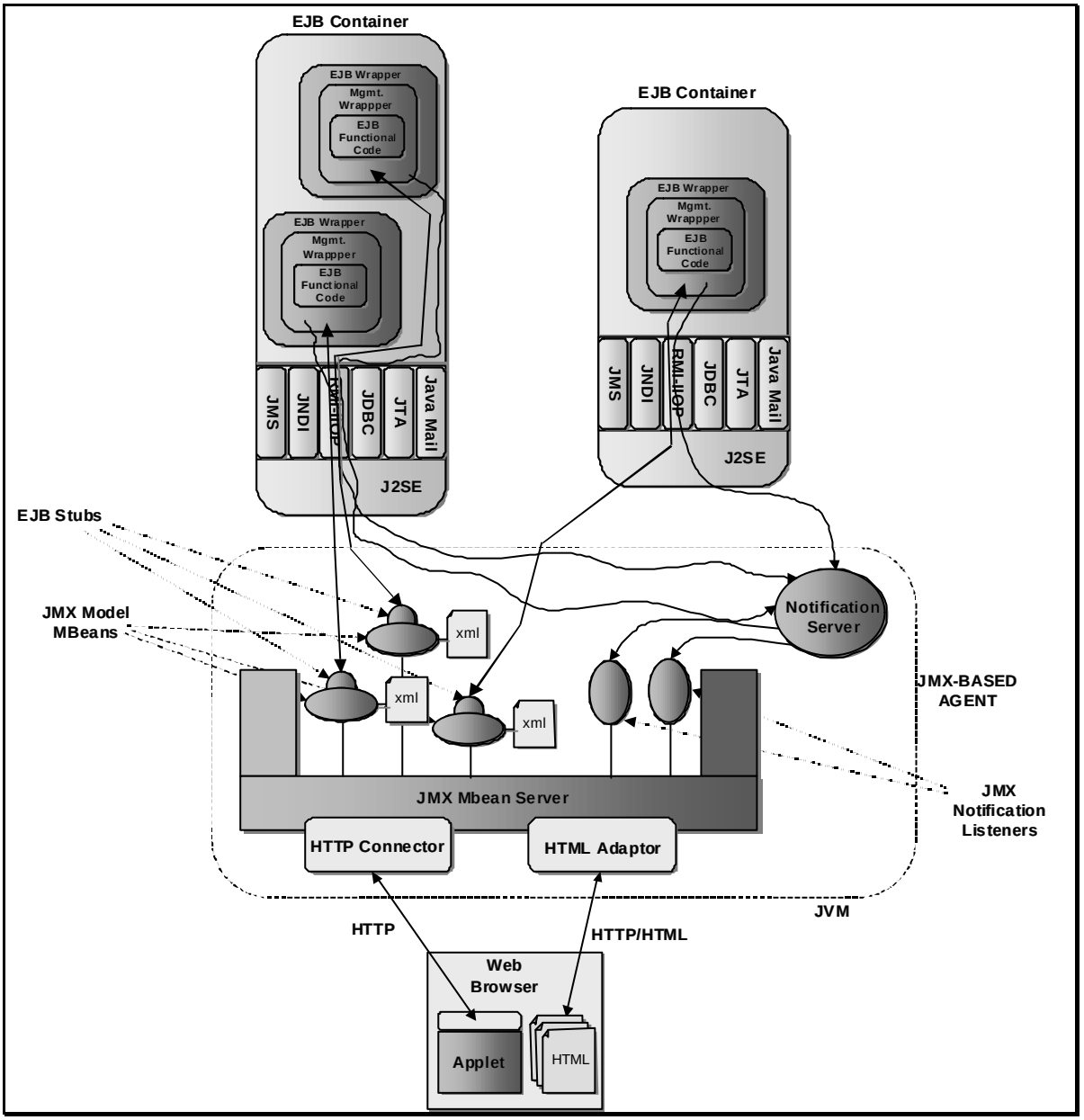

Figure 1 Architecture of the MKBEEM prototype management system

Model MBeans read an XML description file at run-time that describes the management capabilities of a Java resource (in terms of management attributes and operations). That Java resource must be registered within the Model MBean and therefore, when a model MBean

receives an invocation of a particular management operation from the MBean server, it redirects the invocation to the registered Java resource (although it is not a valid MBean) [4].

This approach was selected for the MKBEEM prototype: a Model MBean was instantiated within the JMX MBean Server for each EJB that had to be managed. For each Model MBean an XML management descriptor was created for describing management information of the remote EJB. Furthermore (and that is the most interesting point) a local reference to the Remote Interface of the corresponding EJB was registered into the Model MBean. Therefore, each time a management operation was invoked on the Model MBean, a remote management operation was transparently invoked on the remote EJB.

Thanks to this approach, it is not necessary to develop customised MBeans for each EJB. The JMX-based agent only has to instantiate the appropriate number of Model MBeans, locate the remote EJBs, obtain the references to their remote interfaces and register them within the Model MBeans.

- For collecting notifications generated by the EJBs:

  JMX specifies a complete notification model that includes the definition of *Notifications, Notifications Filters*, *Notifications Broadcasters* (entities that generate *Notifications*) and *Notifications Listeners* (entities that register into *Notifications Broadcasters* in order to receive *Notifications* according to some filtering criteria). This notification model only covers the transmission of events between MBeans within the same JMX agent [9] and, therefore, does not cover the problem of collecting notifications submitted by remote EJBs.

  In order to solve the previous problem, the JMX-based agent for MKBEEM included a "Notification Server" that received JMX notifications remotely (using RMI-IIOP distributed computing capabilities) from the managed EJBs. This RMI-IIOP "Notification Server" implements the "Notification Broadcaster" JMX interface and, therefore, different "Notification Listeners" (with their appropriate Filters) can be registered in the "Notification Server" to receive management information they are interested in.

  A centralised "Notification Server" is used (instead of one server per EJB) in order to simplify the location procedure of that server by the deployed EJBs: the "Notification Server" registers itself at the JNDI server used by the different EJB containers with a predefined name that is known by all the EJBs (that name can be modified provided that the new name is indicated in the appropriate environment property of all the managed EJBs).

  The above support for notifications might be enhanced with the usage of JMS (Java Messaging Service) which recently has become an integral part of the J2EE 1.3 specification

(implementation was not mandatory for J2EE 1.2 platforms). With JMS, the sending of Notifications can be asynchronous [1].

3.2 MANAGEMENT INSTRUMENTATION

The following requirements were taken into account when deciding what management instrumentation scheme to apply to the EJBs of the MKBEEM prototype:

- The instrumentation approach adopted should be generic enough so that it might be applied not only to the MKBEEM prototype but also to any other EJB-based application. This requirement implied that the instrumentation approach should be flexible enough to be able to obtain management information of different types.

- The instrumentation should be as transparent as possible to the developers of the EJBs.

If the main approaches to the instrumentation of "generic" distributed applications (not necessarily EJB-based) are classified according to their degree of generality/flexibility and transparency/automation [3] (as shown in Figure 2) the trade-off between the above requirements might be achieved by "Stub Instrumentation" or "Wrapper-based Instrumentation". Nevertheless, EJB-based applications have an additional characteristic that has to be taken into account when thinking of instrumentation possibilities: after they have been developed, they have to be deployed over the EJB container of a J2EE server. That deployment step implies that the EJB container generates all the necessary stubs and skeletons for RMI-IIOP communications as well as a wrapper for the EJBs containing the support needed for transactional, security, persistence management, etc. related issues.

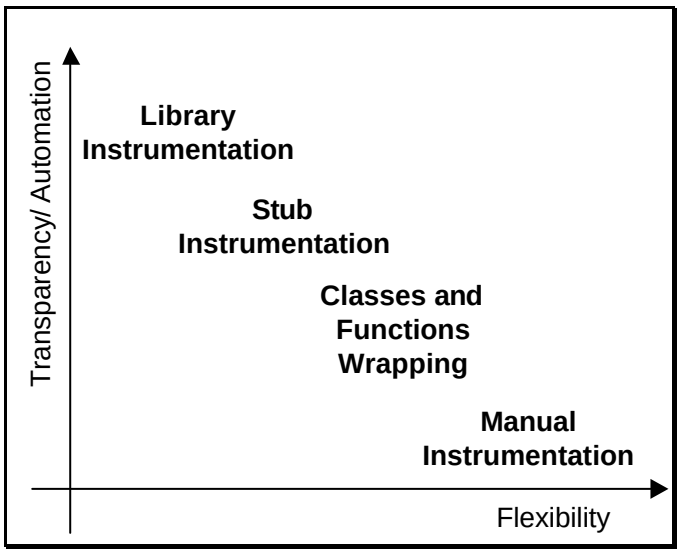

Figure 2 Classification of instrumentation approaches for distributed applications [3]

That means that the degree of transparency and automation of "Classes and Functions Wrapping" might be greater for EJB-based applications than the one suggested in Figure 2 (but maintaining its level of flexibility).

Therefore, the "wrapping" approach was chosen for the instrumentation of the MKBEEM prototype. In this case, the wrappers for the different EJBs had to be coded manually. Using Java class inheritance, the management wrappers could be integrated with the functional code of the application without modifying it.

Figure 3 shows an example of how one particular EJB of the MKBEEM prototype was "wrapped": the `UserAgentEJB` Java class, and the `UserAgent` Java interface have been developed by the functional developers of MKBEEM. `UserAgentWrapperEJB`, `UserAgentWrapper` and `UserAgentHome` constitute the "wrapper" for the `UserAgent` EJB. When a client object invokes a method of the MKBEEM `UserAgent`, the corresponding method of the wrapper is previously invoked. The wrapper is then able, for instance, to send a Notification to the Notification Server of the JMX-based agent (which is implemented by the `EJBJMXControllerInterface` and `EJBJMXControllerImplementation` Java interface and class). According to Figure 3, the communication between the wrapper and the Notification Server is supported by the `EJBJMXBridge` auxiliary class. During this "wrapping" process, the original functional code doesn't have to be modified.

Currently, the JMX-based agent (developed on top of the AdventNet Agent Toolkit 4.1 for Java[6]) has been integrated with the J2EE server (Sun's J2EE Reference Implementation v1.2.1) in the way shown in Figure 1. Several MBeans have been developed in order to collect and process the data generated by the management wrappers of the MKBEEM prototype EJBs. They correlate notifications associated to the invocation of certain functional methods in order to calculate time statistics related to the performance of the MKBEEM prototype. A Web-based manager has been developed using AdventNet Management Builder[7].

4 Conclusions

This paper has presented the ongoing work within the MKBEEM IST project in order to manage EJB-based applications. More precisely, the management architecture (based on the JMX standard) and the management instrumentation of the service-oriented management aspects have been described.

JMX has been chosen as the basis for the management architecture of EJB-based applications due to its modularity (management services are considered as MBeans and therefore can be dynamically

[6] http://www.adventnet.com/producst/javaagent/index.html

[7] http://www.adventnet.com/products/mbuilder/index.html

added an/or removed, as well as managed), and flexibility (different protocol adapters and connectors can be plugged into an MBean server in order to access management functionality and data from other management domains such as SNMP, CIM/WBEM, etc...)

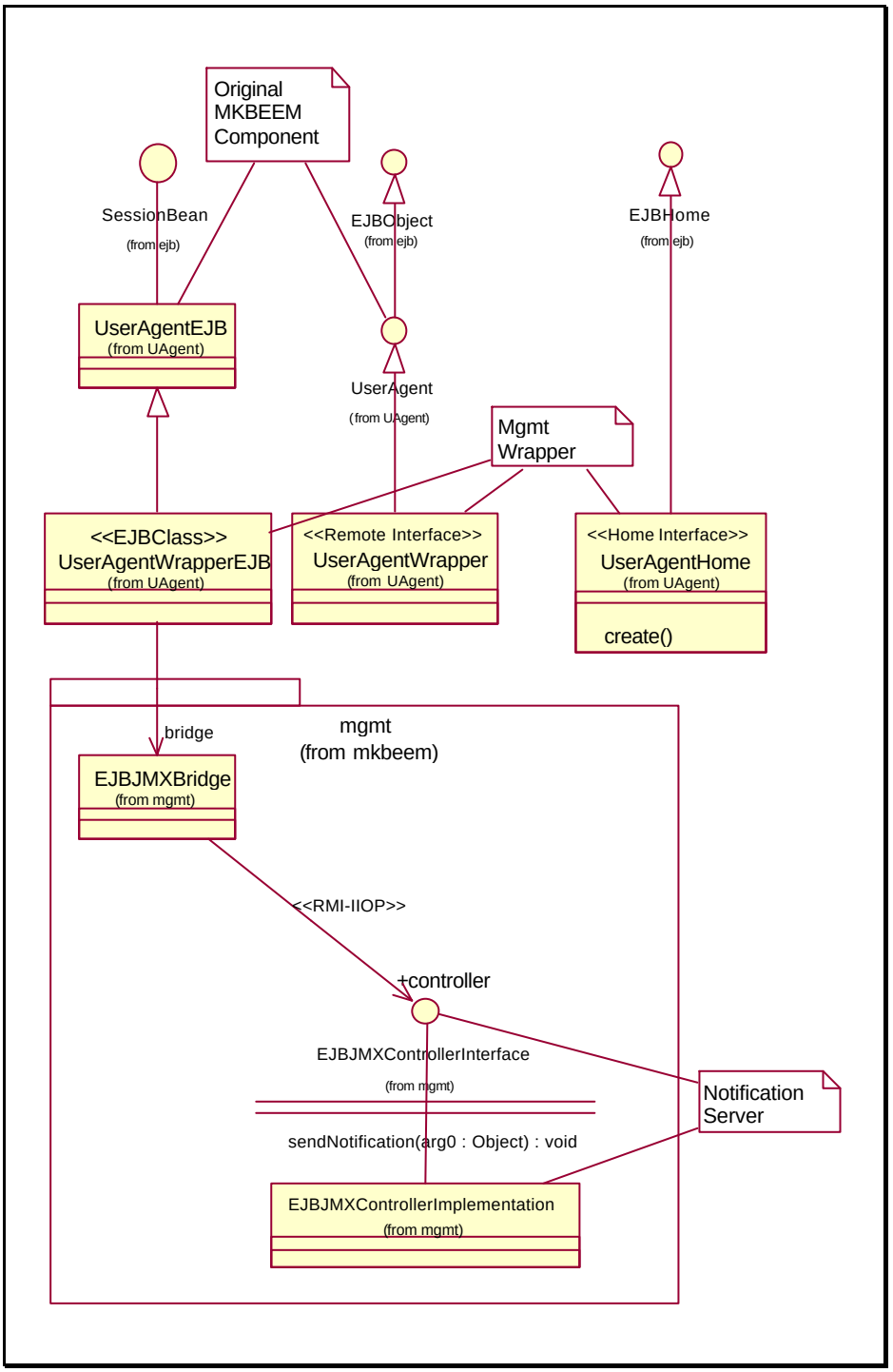

Figure 3 UML class diagram with the design of the wrapper for an EJB of the MKBEEM prototype and its relationship with the Notification Server of the JMX-based agent (see Figure 1)

The described approach to the management instrumentation of the EJB-based MKBEEM prototype (using management wrappers) has the advantage of being transparent to the developers of the

functional aspects of the managed applications while being flexible enough for supporting the main management functionalities.

While the work described in this paper has been mainly devoted to the problem of what might be called “management infrastructure”, future work should be directed towards the automation of management related development tasks. The achievement of this goal would imply:

- Studying how to enhance EJB deployment descriptions with information about the characteristics of the management wrappers (in a similar way as security attributes, transaction properties, etc. are specified)

- Evaluating the alternatives of how J2EE servers might be enhanced with the capability of automatically generating management wrappers according to the deployment information in the same way as current J2EE servers generate wrappers for implementing container-managed persistence, container-managed transactions, etc.

5 References

[1] D. Chappel, G. Pavlik, “Distributed Logging Using the Java Message Service”. Java Developer’s Journal, 6(5). May 2001.

[2] Java Community Process, “J2EE Management”. JSR-77. September 2000.

[3] M.J. Katchabaw, S.L. Howard, H.L. Lutfiyyam, A.D. Marshall, and M.A. Bauer, “Making distributed applications manageable through instrumentation”. In Proceedings of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems (PDSE’97). Boston, Massachusetts, USA. May, 1997.

[4] H. Kreger, “Java Management Extensions for application management”. IBM Systems Journal, 40(1). 2001.

[5] J. Hopkins, “Component Primer”, Communications of the ACM, 43(10). October 2000.

[6] Object Management Group, “CCM FTF Draft of new Chapters”. OMG Document ptc/99-10-04. October 1999.

[7] Sun Microsystems, “Enterprise JavaBeans Specification, Vesion 1.1”. 1999.

[8] Sun Microsystems, “Enterprise JavaBeans Specification, Version 2.0”. Proposed Final Draft 2. April 2000.

[9] Sun Microsystems, “Java Management Extensions Instrumentation and Agent Specification”, v1.0. July, 2000.

[10] Sun Microsystems, “Java Messaging Service”. Version 1.0.2. November, 1999.

[11] Sun Microsystems, “Java 2 Platform, Enterprise Edition Specification, v1.2”. December 1999.

[12] Sun Microsystems, “Java 2 Platform, Enterprise Edition Specification, v1.3”. Proposed Final Draft 3. March 2001.