

# Dynamic instrumentation for the management of EJB-based applications

Nada Almasri , Stéphane Frénot  
CITI, INSA Lyon, Bat Léonard de Vinci  
20, Av. A. Einstein  
69621 Villeurbanne, CEDEX

*nalmasri@telecom.insa-lyon.fr, stephane.frenot@insa-lyon.fr*

## Abstract

*With the growing number of distributed component-based applications for enterprises, an efficient management should be applied over these applications to evaluate their performance, to provide them with a good quality of service(QoS), and to maintain them during their life cycle. In this paper we focus on Enterprise JavaBeans distributed applications. We define a unified and automatic instrumentation mechanism to allow managing EJB-based applications. The mechanism is based on an instrumentation interface extracted automatically from the home and remote interfaces of EJB beans. The interface is then used by the management system of the application server hosting the EJB applications in order to generate Managed objects corresponding to each initial EJB bean of the application. These managed objects are manipulated by a management application through an agent that allows the access of a number of managed objects including those representing the EJBs. In this paper we propose implementing a unified instrumentation for EJB-based applications automatically by integrating an instrumentation service into application servers hosting these applications. The instrumentation service generates a unified instrumentation interface for all EJBs living or willing to live in the application server and then it generates the corresponding Managed objects according to the requirements of the management system used by the application server.*

**Keywords:** EJB, instrumentation, distributed application management, management systems.

## 1. Introduction:

Object-Oriented technologies have had a large impact on industrial information system development process. Object-Oriented features such as encapsulation, composition, and abstraction lead to more flexible and extensible software products. OO programming however has failed to isolate specific standard functionalities in order to be reused in different domains. Component technologies have been introduced to provide isolation and encapsulation to such functionalities and to offer them as services. While the object approach emphasizes design and development, the component approach emphasizes description and deployment [Hal02]. The deployment term is a new convention in applications

implementation which implies the maintenance of the application's components during their life cycle.

Component technologies have, then, expanded to provide distributed systems with distributed component models. Three main distributed models were presented in the industrial world: COM/DCOM from Microsoft [Mic], CORBA Component Model (CCM) from the OMG [OMG], and EJB from Sun Microsystems [EJB01]. There are also some research groups developing component systems in the academic world [NR98,BR00]. All of these models follow the three-tier architecture: presentation tier which is the application's entry point for end users, middle tier where middleware services reside, and the backend tier where data is maintained persistent.

Application servers are middlewares that help isolating the business logic of an enterprise application from the platform-related code. They offer standard services such as security, transaction and persistence for enterprise applications allowing the developers to concentrate on the business logic of an application.

With the growing need for enterprise distributed applications satisfying high performance, scalability and availability features, the demand for effective management, observation, and control solutions has been raised. The management of distributed applications needs additional requirements exceeding what is necessary for stand-alone applications because of the heterogeneity of the operating systems, the heterogeneity of the distributed component models, and the distribution of the manageable code.

Despite of the business application development simplicity brought by the above mentioned distributed component models, the developer still need to implement its own management code to manage its application since no management frameworks were implemented for each of these models. Although few research works handled the distributed application management [KHL99,VAL02,RLR+00], they didn't address the issue of automating the management of the distributed applications even if such applications would follow one of the distributed components models.

Among the different existing distributed component models, we focus in this paper on automating the management of EJB-based applications through instrumentation. The term "instrumentation" was initially used with instruments and devices. The instrumentation meant the control of hardware devices and instruments by

assigning them a soft interface to configure and manage them. The term was basically used by industries, then it was introduced in the research papers in the context of enabling management of different kinds of resources including devices, network services and applications. We can now present our definition of EJB Instrumentation in order to avoid any possible confusion with other possible definitions of “instrumentation”. Considering an EJB application as an instrument, we want to define a way to manage this instrument dynamically. Enabling a dynamic control over EJB applications is what we call EJB Instrumentation. In this paper we provide a unified and automatic instrumentation mechanism to free the developer from the management duties he/she has to add for the EJB applications in order to enable them for management.

The paper is structured as follows. Section 2 presents our motivation. Section 3 provides an overview of instrumentation and application management, we first present the management principles, then the architecture of management system, and finally we present an overview of distributed application management. In section 4 we introduce the EJB instrumentation architecture. In section 5 we present some related work. We finally conclude by summarizing our work and presenting few perspectives in section 6.

## 2. Motivation

Java was the first programming language that presents the notion of WORA (Write Once, Run Anywhere). The success of Java came from this notion that relieved the developers from rewriting the same code for different operating platforms. Since then, all other information technologies were directed to be used in a standard way with different underlying technologies. Likewise, the Java 2 Enterprise Edition (J2EE) [J2EE01] introduced the EJB technology which is a standard way to implement distributed Java applications that are runnable on different application servers. To continue this chain of standardization we need a standard way to manage EJB applications.

Meanwhile most of the J2EE-compliant vendors include ad-hoc management solutions in their products. Some application servers like iPlanet [iPlanet], AppServer [AppServer], WebSphere [WebSphere] handle manageability by adding SNMP (Simple Network Management Protocol) agents [SNMP90]. Other application servers like JBoss [JBoss], BEA Weblogic [Weblogic] and IONA iPortal [iPortal] handle manageability by adding JMX (Java Management eXtensions) agents [JMX00]. In both cases, manageability is limited to the basic services offered by the application server itself like naming, transaction, and security. None of the presented management solutions handled the management of the hosted application functional code. That is why we try in our research works to define a unified way to instrument EJB applications.

We also aim to perform the EJB instrumentation automatically, which frees the application developer from the

heavy management duties. Being both unified and automatic, EJB instrumentation presents several benefits:

**Standardization:** A unified EJB instrumentation allows heterogeneous management systems to access the same instrumented EJB application.

**Management user interface:** performing automatic instrumentation over EJB applications facilitate building management user interface automatically while instrumenting the EJB application without the need to code it manually by the developer.

**Dynamic management:** EJB instrumentation allow dynamic management over EJBs. This functionality provides the management system with flexible management over applications while they are running.

**Fault tolerance:** Enabling dynamic management over EJB-based applications provides them with the ability to repair failures when occurring since the management application will be able to: access the application dynamically, stop the infected component, and reinitialize it with its last operational state.

**Load balancing:** EJB instrumentation enables the management system to collect management information describing the activity of each component of the EJB-based applications. When the system is over loaded with so many client sessions connected to the same station, the management system can migrate one or more application(s) to a more free station according to some load balancing algorithm and redirect clients transparently to that station [AFA02].

## 3. Management and Instrumentation

### 3.1 Principles

*Management* is the mechanism of observing and controlling system elements. Elements might be devices, services, or software applications. Each element in the system needs to be instrumented in order to enable an overall supervisory and control activities to the whole system.

*Instrumentation* is the process of providing a descriptive interface defining the management information and the operating modes of the element in order to enable it for management. Each element of the system is instrumented according to its type and to the required management functions.

*Management systems*, that are responsible to ensure the overall management of the underlying system, use the management information of all the elements in the system in order to provide major management functions like life cycle, configuration, performance, security, and fault management.

*Alarm notifications* are sent by the instrumented elements to the management system when a pre-defined event has occurred. *Events* which are critical for management functions are specified according to a set of *measures* that defines the performance and operating indicators of the system's elements.

According to the type of event received by the management system, it performs pre-set operations in order to

provide the underlying system with the required Quality of Service at a reasonable cost.

### 3.2 Management systems

While some success has been realized in the areas of system and network management, only few efforts have been paid on the management of distributed applications. Despite of the huge existence of different management standards and technologies, they were directed to manage networks and their devices. There are mainly two well known management protocols: The Simple Network Management Protocol (SNMP)[SNMP90] which is an IP-oriented protocol, and the Common Management Information Protocol (CMIP) which is based on the Open System Interconnection (OSI) [ISO91]. Both SNMP and CMIP define management standards, management services and management protocols. This makes them viewed as management frameworks rather than management protocols.

Other management frameworks are also available but are not yet widely adopted and still under improvement, these include: the CIM/WBEM (Common Information Model/Web-Based Enterprise Management) [DMTF01] which is a set of management and Internet standard technologies developed to unify the management of enterprise computing environments. Sun has also defined its own Java specification for the management of Java applications and called it JMX (Java Management eXtensions) [JMX00]. While SNMP and CMIP are network management oriented, CIM/WBEM and JMX are application management oriented and can be dedicated to distributed object computing in order to manage distributed applications.

Despite of the wide variety of management standards[Tay96, MZH98], most management system frameworks follow the Manager/Agent architecture pattern showed in figure 1. The architecture has three component levels:

- **Managed object:** is an abstraction for a device, a service or an application. It exposes the appropriate data in the form of an interface for use by a management system. The data contains both attributes to be monitored and operations that can be performed to control the underlying resource.
- **Agent:** is the responsible of the direct interaction with the managed resource. It is the only component capable of performing management operations exposed by the managed resource. The agent provides some other management services to be used by management applications like the scheduling service and the event notification service.
- **Manager:** is the application responsible of managing one or more resources on a network. It communicates with the agent to collect management information and to perform management tasks on the Managed resources associated with the agent.

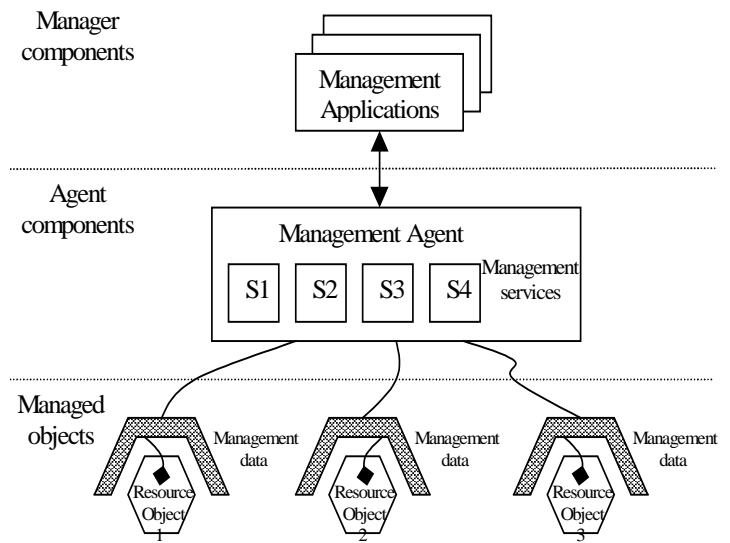


Figure 1: Manager/Agent architecture model

In our research work we apply this architecture to manage distributed applications. We focus mainly on the management of J2EE applications and more precisely on the instrumentation of EJB applications. Our work consists mainly on automating the transformation of an EJB bean into a managed object. We then work at the first level of the above described architecture.

### 3.3 Distributed Applications Management

With the growing number of complex distributed applications having a large number of components, the need for effective management over the whole system is increasingly important. Unlike stand-alone applications, distributed applications are mission critical since they are expected to serve many client sessions all at the same time and are expected to be continuously available for clients. This would require applying management over all the components, their behavior, their functionality, and their life cycle in order to manage their creation, suspension and reactivation. High manageability of distributed applications is then very essential for enterprise applications to provide their clients with a high quality of service.

Most of the work on quality of service has concentrated on the network and communications infrastructure. Recently, QoS has also become increasingly important in distributed applications due to the increasing use of internet which facilitates the access to the distributed applications. Many efforts were paid on performance, monitoring, and tests of distributed system [LSZ+98]; some of them were applied on CORBA distributed applications [SKK+00] and others were applied on EJB applications [MMP01, PPM99]. Although no standard parameters for the QoS of EJB applications are defined yet, the QoS should be taken into consideration when developing EJB applications which would essentially implies effective management over EJB applications.

Until now, management tends to be developed specifically for one application since different applications have different specialties, different structures, and different goals. Nevertheless, with the arrival of the component standards (CCM, COM+, EJB), distributed applications following the same model could be managed in a standard way using one of the existing management frameworks [SNMP90, ISO91, DMTF01, BCS00] since none of these component models defines its own management framework yet.

#### 4. EJB Instrumentation Architecture:

In this section we define an architecture to instrument EJB applications dynamically and automatically in a unified way. The architecture doesn't depend on a specific application server or on a specific management framework so that it can be applied for different application servers with different management systems.

An EJB application contains a number of EJB beans packaged together in a Jar file with an XML descriptor that contains all information about the EJB beans characteristics and their usage of the services provided by the application server. The main services provided to EJB applications by their hosting application servers are: Transaction, Security, Persistence, Messaging, and Naming. Some application server vendors provide additional services such as an administration service to monitor the activities of the application server. Although this administration service provides a kind of management over the EJB's containers, they don't handle the instrumentation of the EJB functional code that represents the business behavior of an application.

Our instrumentation architecture is based on integrating a new instrumentation service beside the existing ones to provide all EJB beans in an EJB-based application with an instrumentation interface that can be used later by different management systems in order to build the corresponding managed objects.

The instrumentation interface is a description of how an EJB can be managed. Some research works [STK96] proposed an object management interface description that extends the interface description language known from CORBA and DME (Distributed Management Environment) compliant systems [OSF92]. In our instrumentation architecture we followed the same interface description with XML as a description language.

In the rest of this section we first present our view of a managed object representing an EJB (we will call it managed EJB for short), then we describe the parts of the EJB instrumentation interface, and finally we introduce the architecture of the instrumentation service.

##### 4.1 Managed EJB

Each component of a distributed application should be a managed component in order to enable the whole application for management. Thus to manage an EJB-based application, each of its EJB beans should be transformed into a managed

component that we call managed EJB. The implementation of the managed EJB depends on the management system. In the CIM management system, a logical element is managed by implementing the CIM\_LogicalElement, while in the JMX management system a managed objects is implemented as a managed bean or MBean. The main problem that rises in application instrumentation is defining the parts of the code that need to be instrumented in the application. This process often requires human intervention in order to choose the critical parts of an object to be instrumented. While some methods would be necessary to automate the placement of instrumentation code in a large number of applications, EJB applications are already structured in a way that the different parts of the code are well defined. This allows the selection of instrumentation parts automatically and in a unified way for all EJBs. Home and Remote interfaces of an EJB bean defines the important methods of an EJB to be managed.

The home interface defines the bean's life cycle methods: creating new beans, removing beans, and finding beans. These methods can then be used to manage the life cycle behavior of the EJB bean. The remote interface defines the bean's business methods: the methods that the bean presents to the outside world to do its work. These methods can be used to manage the business behavior of the EJB bean. Thus, both life cycle methods and business methods should be presented in the instrumentation interface since they play an essential role in the management of EJBs. Figure 2 shows the abstract architecture of the Managed EJB.

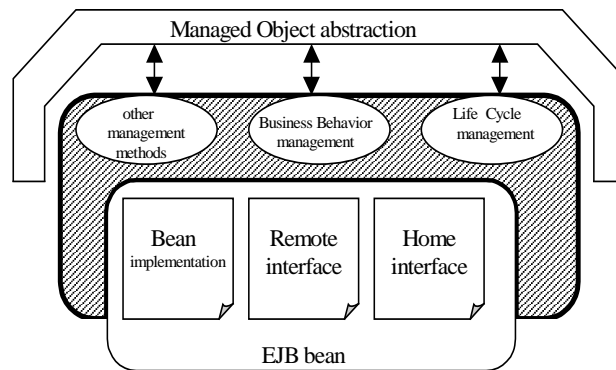


Figure 2: Abstract architecture of a managed EJB

##### 4.2 EJB Instrumentation interface

One of the most time consuming challenges in information technology has been to represent data in a standard way so that it can be manipulated by and exchanged between heterogeneous information systems [APK01]. Presenting the data in a neutral specification language such as XML can greatly reduce this complexity and create data that can be read by different types of applications. Thus, to create the instrumentation interface we chose to use the XML language that has proven its strength and flexibility to define standards in different domains [Hal99]. Each management system can then access this interface, parse it, and create its

appropriate managed object describing the underlying EJB. Any XML document can be transformed automatically to another document form conforming to the management system using the XSLT [XSLT] which uses an XML parser to transform the XML document into any other document form according to a style sheet transformation document.

The EJB instrumentation interface is then written in XML, and is based on a previous proposed object instrumentation interface that consists of three elements: attributes, operations, and states[STK96]. We followed this interface description after including few modifications to adapt it to EJB beans. Dealing with Java beans, object's attributes are not accessed directly since they have their corresponding getter and setter methods. Thus we gathered both attributes and operations under one management category, the business behavior management category. The object states represent the life cycle management category. Beside these two management categories we added a third one that is appropriate to EJB-based applications, the EJB QoS management category.

**Life cycle Behavior management:**

The first task of EJB Instrumentation is to manage and monitor the life cycle for each EJB instance. This management category allows dynamic instantiation, deletion and locating of EJB objects and can be used to perform some statistics on instances in their different states which allows performance testing of the EJB application, the application server, and the configuration specified by the EJB application developer. Life cycle behavior management methods are extracted from the home interface of the EJB bean.

**Business Behavior management:**

The second task of EJB instrumentation is to manage the business behavior of an EJB bean by managing all of its business methods. Managing these methods would allow us to test the time needed to have a response from the method, the number of times a method is called, and other management information depending on the requirements of the manager application. Business behavior management methods are extracted from the remote interface of the EJB bean.

**QoS methods:**

The QoS methods allow a global management over the EJB application. They allow testing the pre-set parameters to ensure a specific quality of service. In contrast to both business and life cycle behavior methods that depend on the implementation of the EJB, QoS methods are independent on the implementation of the EJB beans; that is, they are applicable to all EJB beans. The goal of these methods is to gather information about the performance properties of the application in order to perform the appropriate actions to provide a better quality. These methods may include: getting the number of instances for each EJB bean of the EJB application, getting the most active EJB beans, allowing EJB migration, and others.

This instrumentation interface is generated automatically from the EJB code thanks to the direct relation between the main parts of an EJB bean and the above mentioned three management categories. The generation is done automatically by the EJB instrumentation service.

**4.3 EJB instrumentation service:**

To allow EJB instrumentation for all EJB applications deployed in an application server we need to integrate a new instrumentation service beside the standard services offered by all application servers. The main task of the EJB instrumentation service is to create automatically the instrumentation interface for each EJB bean of a new deployed EJB-application in order to implement its corresponding managed EJB according to the directions supplied by the management system with a style sheet. The StyleSheet file describes how to transform the XML instrumentation interface into a managed EJB that conforms to the specifications of the managed objects maintained by the management system. The managed EJB represents the managed object level of a management framework. For example, for a system managed by JMX management framework the instrumentation service should implement the managed EJB as an MBean. However, in a system managed conforming to the CIM model the instrumentation service should implement the managed EJB as a CIM\_LogicalElement object.

The instrumentation service will generate the managed EJB automatically from the extracted XML instrumentation interface and according to the supplied StyleSheet. There will be as many managed EJBs as there are EJB beans in the application. Figure 3 shows the architecture of the proposed EJB instrumentation.

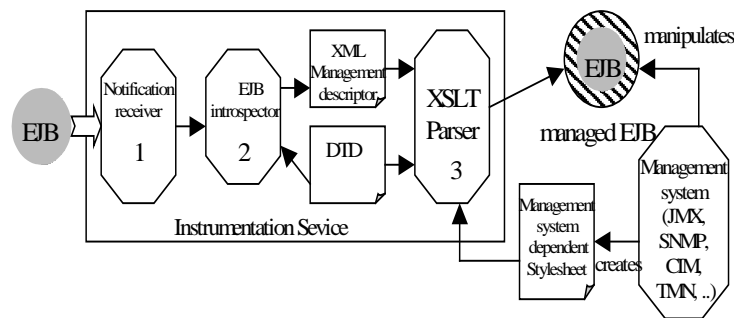


Figure 3: EJB instrumentation architecture

The EJB receiver (1) is responsible of receiving new deployed EJB applications. For each EJB bean of the application the notification receiver retrieves both the home and remote interfaces and sends them to the EJB introspector (2) which generates the corresponding XML instrumentation descriptor that conforms to the EJB instrumentation DTD.

The XSLT parser (3) is the last part of the instrumentation service. It allows different management systems to supply the StyleSheet file where they describe how

to transform XML instrumentation interface into a managed object. The generation of managed objects is done dynamically at run-time when a new EJB is deployed.

The produced managed EJB can be seen as a wrapper of the managed parts of the EJB. It receives the method invocations for the managed methods and redirect them to the underlying EJB.

#### 4.4 Example : HelloWorld managed EJB:

To validate our proposed architecture we implemented the instrumentation service on the JBoss application server that uses JMX as its management framework. We created a StyleSheet file describing the transformation of the XML instrumentation interface into an MBean (the managed object of the JMX management system). The instrumentation service generates an XML instrumentation interface for each EJB. Then it generates the corresponding managed EJBs according to the supplied Stylesheet. The managed EJBs are implemented as MBeans to conform to the JMX specification. The MBeanServer of the JMX framework will then register these new MBeans so that they can be consulted by other existing MBeans or some management applications.

To show the steps that an EJB bean passes through in our management framework we present HelloWorld EJB example. We deploy the EJB on the application server where we have integrated the EJB instrumentation service. Figures 4(a,b,c) show the three main classes of the EJB bean.

```
public class HelloWorldBean implements SessionBean {
    String message;
    public void ejbCreate(String message){
        this.message=message;
    }
    public String getHelloMessage() {
        return this.message;
    }
    public void ejbPostCreate(){}
    public void ejbRemove(){}
    public void ejbActivate(){}
    public void ejbPassivate(){}
    public void setSessionContext(SessionContext sc){}
}
```

4.a HelloWorldBean implementation

```
public interface HelloWorldHome extends EJBHome{
    HelloWorld create(String message) throws RemoteException,
    CreateException;
}
```

4.b HelloWorld Home interface

```
public interface HelloWorld extends EJBObject{
    public String getHelloMessage() throws
    RemoteException;
}
```

4.c HelloWorld Remote interface

After the introspector phase of the instrumentation service the XML instrumentation descriptor showed in figure 5 will be generated.

```
<?xml version="1.0" ?>
<!DOCTYPE MBean SYSTEM "MBean.dtd">
<MBean name="HelloWorld">
<BusinessMethods>
    <method name="getHelloMessage" id="0"
    abstract="false" static="false" synchronized="false"
    volatile="false" transient="false">
        <type>String</type>
        <throws />
    </method>
</BusinessMethods>
<LifecycleMethods>
...
</LifecycleMethods>
<QoSMethods>
...
</QoSMethods>
</MBean>
```

Figure 5. HelloWorld XML instrumentation descriptor

The XSLT parser will take the above XML instrumentation descriptor and produce, corresponding to the supplied Style sheet, the appropriate MBean representing this EJB.

Figures 6.a and 6.b show the two main classes of the managed EJB.

```
public class InstrumentedHelloWorld implements
InstrumentedHelloWorldMBean{
    Vector beanInstances=new Vector();
    //Life Cycle Methods
    public String create(String msg){
        Context ctx=new InitialContext();
        Object ref = ctx.lookup("HelloWorldBean");
        HelloWorldHome home = (HelloWorldHome)
        PortableRemoteObject.narrow(ref, HelloWorldHome.class);
        HelloWorld instance=
            beanInstances.addElement(home.create(msg));
        newInstCreated(instance);
        return "Instance "+beanInstances.size();
    }
    //business methods
    public String getHelloMessage(){
        HelloWorld requiredInst=getInstance();
        getHelloMessageInvoked(requiredInst);
        return requiredInst.getHelloMessage();
    }
    //QoS methods
    public void newInstCreated(){...}
    public void getHelloMessageInvoked(){...}
    public int getActiveInstanceNumber(){...}
}
```

Figure 6.a: InstrumentedHelloWorld MBean class

```

public interface InstrumentedHelloWorldMBean{
    public String create(String msg);
    public String getHelloMessage();
    public int getActiveInstanceNumber();
}

```

Figure 6.b: InstrumentedHelloWorld MBean interface

## 5. Related work

Similar studies were presented in the past by a research group at the university of western Ontario in Canada [KHL97, HGB95, VLM+98]. They were directed to manage distributed applications through instrumentation, but they considered that management should be added to applications while they are being developed which would make it heavier to the developer to implement the application. These studies were directed to object approaches while our work is directed to component approaches.

Another research group in Spain is currently working on an European project [MKB] devoted to the management of EJB-based applications. Some of their experiences in the management of EJB-based applications were presented in Germany [AVL+01]. In fact, they instrument EJBs by wrapping them manually into Model MBeans of the JMX management framework.

Finally, Sun is now developing the Java 2 Enterprise Edition Management Specification [JSR77] that seems to be implemented based on JMX framework. The specification defines a common framework for the delivery of Java enterprise management and monitoring services. In addition, it handles the mapping to both SNMP and CIM/WBEM. The specification is not complete and until now it doesn't seem to handle automatic instrumentation although it defines MEJB which is itself built as an EJB component. We are still waiting this Management framework to combine our instrumentation solution with it since we believe they do complement each other.

## 6. Conclusion and perspectives

The work we describe here is a part of an ongoing research works on EJB-based distributed application management and services integration in application servers. In this paper we propose instrumenting deployed EJB applications by integrating a new instrumentation service to the application server. The instrumentation service generates an XML instrumentation interface describing the management tasks of the EJB. XML is used to standardize the management view of EJB applications regardless to the management system. The implementation phase of the managed EJB is done by an XSLT parser that takes a stylesheet file describing the transformation of the XML file into a managed object following the specification of a particular management system. This implies that the XML should contain the minimum information required by the variety of the management systems specified in section 3.2.

This can be realizable since most management information are common to these systems. In fact many transformation techniques exist between different management system [FES99, JSR146, DMTF97]. Having a common management interface would greatly facilitate the transformation between management systems.

To validate our approach we implemented the instrumentation service under the JBoss application server and test it with the JMX management system. Although we didn't test it with other management systems, we think that the management information contained in the XML instrumentation interface are sufficient for other management systems since there exist APIs to map JMX to both CIM and SNMP. In future works we will study the architecture and the component of an ideal XML instrumentation interface that would present all necessary management information without having too much unused information.

The instrumentation that we perform over EJBs allows the management of EJB-based applications without changing the application's code, and without considering management at development time. This would free the developer from additional work which is considered as standard functionality for all EJB-based applications since they all follow the same development logic.

EJB instrumentation is the first step toward managing EJB-based applications. Manager applications for EJB-based applications that will use the managed EJBs should also be studied in order to perform global management over one or more EJB-based applications.

## References

- [AFA02] S. Frénot, M. Avin Sotres, N. Almasri. *EJB components Migration Service and Automatic Deployment*. Rapport de recherche RR-4480 INRIA, juin 2002
- [APK01] A. Anagnostaki, S. Pavlopoulos, D. Koutsouris. *XML and the Vital Standard: the document-oriented approach for open telemedicine applications*. Proc. MedInfo 2001 V. Patel & al. (ed.) Amsterdam: IOS press, 2001.
- [AppServer] <http://www.borland.com/bes/appserver/>
- [AVL+01] J.I. Asensio, V.A. Villagra, J.E. LopezDeVergara, R. Pignaton, J.J. Berrocal. *Experiences in the management of an EJB-based e-commerce application*. Proceedings of the 8th HP OpenView University Association Plenary workshop (HPOVUA'01), Berlin, Germany, 2001.
- [BCS00] P. Bellavista, A. Corradi, C. Stefanelli. *An Integrated Management Environment for Network Resources and Services*. IEEE Journal on Selected Areas in Communication, Vol. 18, No. 5, May 2000.
- [BR00] E. Bruneton, M. Riveill. Javapod. *An Adaptable and Extensible Component Platform*. Workshop on Reflective Middleware, April 2000. New York USA.
- [DMTF97] Distributed Management Task Force (DMTF). *DMI to SNMP Mapping Standard*. 1997. <http://www.dmtf.org/standards/snmp.php>

- [DMTF01] Distributed Management Task Force (DMTF). *WBEM: Web-Based Enterprise Management*. 2001. [http://www.dmtf.org/standards/standard\\_wbem.php/](http://www.dmtf.org/standards/standard_wbem.php/)
- [Dum99] E. Dumbill. XML Inter-Application Protocols. <http://www.xml.com/pub/a/1999/10/open/index.html>.
- [EJB01] Sun Microsystems. *EJB Specification V2*. August 2001. <http://java.sun.com/products/ejb/>
- [FES99] O. Festor, P. Festor, L., N. Ben Youssef. *Integration of WBEM-based Management Agents in the OSI Framework*. Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management (IM'99), Boston, Massachusetts, U.S.A. May 1999.
- [Hal02] S.D. Halloway. *Component Development of the Java Platform*. Addison-Wesley, 2002.
- [HGB95] J.W. Hong, G.W. Gee, M. A. Bauer. *Towards Automating Instrumentation of Systems and Applications for Management*. Proceedings of the 1995 IEEE Global Telecommunications conference, Singapore, Novembre 1995.
- [iPlanet] <http://docs.ipplanet.com/docs/manuals/ias.html>
- [iPortal] [http://www.iona.com/products/ip\\_ipas\\_home.htm](http://www.iona.com/products/ip_ipas_home.htm)
- [ISO91] ISO. Information Processing Systems- Open Systems Interconnection. *Common Management Information Protocol Specification(CMIP)*. International Organization for Standardization, International Standard 9596-1, 1991.
- [J2EE01] Sun Microsystems. *J2EE Specification*, V1.3, July 2001. <http://java.sun.com/j2ee/>.
- [JBoss] <http://www.JBoss.org>
- [JMX00] Sun Microsystems. *Java Management eXtensions*. July 2000, <http://java.sun.com/products/JavaManagement/>
- [JSR77] Java Community Process. *J2EE management specification*. 2001. <http://jcp.org/jsr/detail/077.jsp>
- [JSR146] Java Community Process. *WBEM Services: JMX Provider Protocol Adapter*. <http://jcp.org/jsr/detail/146.jsp>
- [KHL97] M.J. Katchabaw, S.L.Howard, H.L. Lutfiyya, M.A. Bauer. *Making Distributed Applications Manageable Through Instrumentation*. IFIP/IEEE International Symposium on Integrated Network Management (IM '97), (San Diego, California), May 1997.
- [LSZ+98] J.P. Loyall , R.E. Schantz, J.A. Zinky, D.E. Bakken. *Specifying and Measuring Quality of Service in Distributed Object Systems*. Proceedings of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98), 20-22 April 1998, Kyoto, Japan.
- [MIC] Microsoft corporation. *COM/DCOM technologies*. <http://www.microsoft.com/com>.
- [MKB] MKBEEM : *Multilingual Knowledge Based European Electronic Marketplace*. <http://mkbeem.elibel.tm.fr/>
- [MMP01] A. Mos and J. Murphy. *Performance Monitoring of Java Component-Oriented Distributed Applications*. Proceedings of 9th IEEE Conference on Software Telecommunications and Computer Networks (SoftCOM), October 2001.
- [MZH98] J-P Martin-Flatin, S. Znaty and J-P Hubaux. *A Survey of Distributed Network and Systems Management Paradigms*. Ecole Polytechnique Federale de Lausanne EPFL, Technical Report SSC, Switzerland, Aug 1998.
- [NR98] R. Natarajan and D.S. Rosenblum. *Merging Component Models and Architectural Styles*. Proceedings of the Third International Software Architecture Workshop, Lake Buena Vista, FL, November 1998. pages 109-111.
- [OSF92] OSF. *The OSF Distributed Management Environment (DME) Architecture*. Open Software Foundation, Cambridge MA, May 1992.
- [PPM99] R. Pospisil, M. Prochazka, V. Mencl. *On Performance of Enterprise JavaBeans*, Presented at the Objekty'99 conference, Prague, Nov 1999.
- [RLR+00] G. Rackl, M. Lindermeier, M. Rudorfer, B. Suss. *MIMO: An Infrastructure for Monitoring and Managing Distributed Middleware Environments*. Middleware 2000 IFIP/ACM International Conference on Distributed Systems Platforms, volume 1795 of Lecture Notes in Computer Science, pages 71-87. Springer, April 2000.
- [SKK+00] D.C. Schmidt, V. Kachroo, Y. Krisnamurthy, F. Kuhns. *Developing Next-generation Distributed Applications with QoS-enabled DPE Middleware*. IEEE Communications magazine, Vol 17, No. 10, October, 2000.
- [SNMP90] J.D. Case, M. Fedor, M.L. Schoffstall, and C. Devin. *Simple Network Management Protocol (SNMP)*. The Internet Engineering Task Force, May 1990. Request for Comments 1157.
- [STK96] A. Schade, P. Trommler, M. Kaiserswerth. *Object Instrumentation for Distributed Applications Management*. In: Proceedings of the IFIP/IEEE International Conference on Distributed Platforms ICDP'96, Dresden, Germany. 1996.
- [Tay96] S.D. Taylor. *Distributed Systems Management Architectures*. Master thesis in Computer Science, University of Waterloo, Canada 1996.
- [VAL+02] V. A. Villagra, J.I. Asensio, J.E. LopezDeVergara, J. J. Berrocal, R. Pignaton. *An approach to the transparent management instrumentation of distributed applications* Approved for its publication as a short paper in the Proceedings of the IEEE/IFIP NOMS 2002 Network Operations and Management Symposium, April 2002. Florence, Italy.
- [VLM+98] R. Versteegh, H. Lutfiyya, A. Marshall, M. Bauer. *Support for Distributed Systems Management Application Development*. Ninth Annual IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM98), October 1998.
- [XSLT] W3C. XSLT transformations. November 1999.
- [WebSphere] <http://www.ibm.com/websphere>
- [Weblogic] <http://www.bea.com/products/weblogic/server>