Enhancing the Management of a J2EE Application Server using a Component-Based Architecture

Takoua Abdellatif
Bull SA
Sardes Project, LSR-IMAG / INRIA
655, Avenue de l’Europe, 38330 Montbonnot, France
Takoua.Abdellatif@inrialpes.fr

Abstract

In this paper we describe our work on improved management of middleware. We argue that a component-based approach to design of middleware results in greatly enhanced and easily extensible management capabilities. As a proof of concept, we present our re-engineered, component-based version of JOnAS, an open-source J2EE server. We describe how we have redesigned and reimplemented JOnAS using Fractal, a novel component model that provides flexible control capabilities and hierarchical composition. Next we show how such adaptation allows us to enhance the management functionalities, and in particular how it allows building management systems for both centralized and distributed environments. We mainly focus on the configuration and deployment functionalities we have developed for J2EE servers in a cluster environment.

1. Introduction

The Sun™ J2EE specification [1] defines the design of J2EE applications as components (EJBs and JSP/Servlets). The goal is to facilitate and standardize the development, deployment, and assembling of application components. Application components are deployable on the J2EE application server (AS) middleware. The middleware contains a set of services for naming, messaging, transactions, persistence, security, logging, and so on. However, the specification does not impose any implementation model in the construction of the middleware. This is left to the AS providers. In this paper, we show that adopting a component-based architecture of the middleware improves the management functions. Indeed, a management based on the knowledge of the system architecture allows to handle the inter-dependence between the system components and enables performing reconfiguration tasks.

In this paper, we reconsider the architecture of JOnAS, an open source application server. We adopt Fractal [5] component model to implement the AS services. Fractal model allows to build modular component-based systems where the relations between the components are explicitly expressed with local (in a same JVM) or remote bindings and hierarchical encapsulations. Consequently, by encapsulating the AS service in a Fractal component, the service becomes an independent unit of deployment and configuration and the relations between services are made explicit. This allows more fine-grained management functionalities. For example, it is possible to reboot a faulty service instead of rebooting the whole application server. Furthermore, we use Fractal model to encapsulate and isolate a set of resources in the distributed environment to apply some common management policies. We qualify our AS prototype as “à la carte”1 because it offers an on-demand service deployment and configuration following a system architecture description. In the rest of the paper, we call jonasALaCarte our AS prototype.

This paper is structured as follows. Section 2 presents JOnAS application server architecture and JOnAS management in a single JVM and in a distributed environment taking as an example the cluster environment. We introduce Fractal model in section 3 and the reason for this choice. Section 4 presents jonasALaCarte architecture, ou re-engineering work and how we can improve management features thanks to a component-based approach. Section 5 presents related works in improving J2EE management and comparison with other open source AS. We end the paper with a

---

1 “À la carte” is a French expression to qualify on-demand services.
conclusion and the presentation of the perspectives of our work.

2. JOnAS application server

JOnAS is an open source application server certified as J2EE compliant (it passed the Sun Compatibility Test Suite for J2EE 1.4). JOnAS is developed within ObjectWeb Consortium [12]; the server embeds different Software from different ObjectWeb members. Indeed, the services that offer the non-functional properties to the J2EE applications encapsulate software from different developers. This involves challenging management needs to handle this heterogeneity, particularly handling the versioning issues.

The distribution aspect of the AS are not addressed by J2EE standards. JOnAS proposes some ad-hoc solutions to manage a set of instances on different JVMs. In this section, we briefly describe JOnAS architecture in its standalone configuration and in a cluster (as example of a distributed environment). We summarize in the last part of this section, the current management drawbacks and the requirements we target to fulfill with the component-based architecture of jonasALaCarte.

2.1. JOnAS architecture

The JOnAS architecture, as illustrated in figure 1, is a set of services and management classes. For consistency, only part of the services is present in the figure. The EJB container is encapsulated in an EJB service and represents an execution environment for the EJB components. The Web service is in charge of running a Servlet/JSP Engine. Currently, this service can be configured to use Tomcat [3] or Jetty [13]. Both the EJB and the Web services need to use the Registry service. The latter provides the JNDI API to services and application components to bind and lookup remote objects (e.g. EJB Homes) and resource references (JDBC data source, Mail and IMS connection factories, etc.). The Transaction service encapsulates a Java Transaction Monitor called JOTM [14] and provides transaction management for EJB components as defined in their deployment descriptors. The IMX service allows exposing the resources in JOnAS following the information model of JSR77 [7]. Thus, all services need to communicate with IMX service to register their respective resources. JOnAS offers also a DB service that wraps a Java database.

JOnAS has two major drawbacks: first, relations between services are implicit. This is due to the fact that services do not communicate using well defined interfaces. Second, the life cycle of services is the same than the one of the application server: the Service manager component simultaneously starts and stops all the services.

2.2. JOnAS in a cluster environment

J2EE clusters currently draw industrial and researchers’ interest and still raise challenging management issues.

Figure 2. JOnAS in a cluster environment

Figure 2 presents a typical architecture of a clustered application server, adopted in JOnAS clusters. The goal is to offer load balancing, high availability, and failover properties by replicating the Web containers at the Web tier and the EJB containers at the EJB tier. The plug-in Apache Tomcat mod_jk dispatches HTTP requests from the Apache Web server to Tomcat instances running as JOnAS Web containers. HTTP in-memory session-replication mechanism is implemented in Tomcat to ensure coherence between replicas. A special Registry service is used for load balancing the requests between the
2.3. Management requirements

Managing a J2EE application server implies managing two parts: the applications and the middleware. The application management is widely addressed by the J2EE specifications. We are interested here in the middleware part whose architecture is left open to the AS providers. The management of the middleware implies configuring and deploying the services, monitoring the different middleware parameters to detect the anomalies and adapting the system when external events occur like failures and performance variations. Examples of adaptation are rebooting part or the whole system if a failure is detected, adding a Web and/or EJB container replica in a cluster if the number of client requests increases and changing a service implementation configuration (or version) at run-time.

In JOnAS, the JMX service allows to monitor the system following the JSR77 standard. The manager can be notified when failures occur and can get regularly a set of performance indicators like CPU usage, the number of threads, the number of objects in the different pools etc. JOnAS offers also a way to configure the services with configuration files. However, currently, the deployment operation is only possible at the granularity of the AS which means that the services are deployed during the AS deployment time.

Furthermore, although the service implementation expose a start/stop interface, since the dependencies between services are implicit, it is not possible currently to restart a service when the AS is running. Indeed, we don’t control currently the side-effect on the other services when rebooting a service. Consequently, it is not possible to perform adaptation operations at the granularity of the service. It is not possible for example, to reboot a faulty service without rebooting the whole AS, or to move a service from a saturated machine to an idle one without deploying a new AS. Similarly, service upgrade and versioning at run-time are not possible without necessarily stopping the server. In a cluster environment, the need for a management at the granularity of a service is also raised. In this distributed environment, we need a more selective replication mechanism. EJB and Web services are generally the bottleneck, so we need more replicas of these services than the Registry service for example. However, a replication of a service means currently the replication of the whole AS and more management tools are needed to ensure the consistency between the AS replicas configurations. We actually need that the service becomes the unit of replication and the configuration can potentially be centralized.

To have an adaptation at the granularity of a service, services have to be configurable and deployable separately. Each service needs to have its own life cycle independent of the AS one. Moreover, since modifying a part of a system at run-time may involve modifications in the other parts that depend on it, the AS also requires identifying the dependencies between the services.

In addition, in a distributed environment, as a cluster, new levels of management abstractions are needed. For example, it is interesting to expose the Web tier, which contains a set of Web containers, as a single virtual Web container to the manager to aggregate and simplify some administration tasks. For example, instead of deploying a same application on each Web container replica, the manager needs a simpler operation: deploying the application on the tier. This hides the distribution and the modifications in the tier that will be handled at a lower level. Furthermore, a J2EE cluster can run different J2EE applications. It is important to group the resources used by an application under a management unit to control the application performance separately.

More generally, we need a tool to group together a set of entities having common properties in order to define common management policies and common management operations. This tool is generally called domain [18] in the management area. Domains are used in different contexts and for different goals: a domain of services for fine-grained adaptation to domains for management aggregation and isolation. We identified the need to express explicitly the relations between the services and the need for the encapsulation feature of the domains. For less complexity, we need to have a uniform representation for all these management units. The model that fulfills these requirements in term of architecture description (explicit relations and encapsulation) is Fractal as we explain it in the next section.
3. Fractal component model

This section introduces Fractal [4] features and justifies why it is the appropriate model to solve our management requirements. Fractal component model defines two kinds of components: primitive components and composite ones. Composite components encapsulate a group of components which allows dealing with them as a unique entity.

Fractal components communicate using explicit bindings. Binding two components means joining two interfaces with connectors (Java references, RMI or through other Fractal components).

Finally, Fractal components can have an arbitrary number of controllers. Controllers are meta-objects allowing the control of the component they belong to. The Fractal library contains several controllers: the content controller allows listing, adding, and removing subcomponents in the parent component; the Life-cycle controller allows starting and stopping the component; the Attribute controller allows setting and getting configuration attributes.

![Figure 3. Architecture of a Fractal Component](image)

Figure 3 illustrates the different constructs in a typical Fractal component architecture. The thick boxes denote the controller part of the components. Arrows correspond to bindings; the interfaces appearing on the top of a component represent the controllers, the interfaces on the left are server interfaces and on the right are client interfaces.

The architecture of Fractal components can be defined in an ADL [15] (Architecture Description Language) file where composition, bindings and controllers are expressed in XML-like syntax. It is also possible to specify on which node or JVM a component has to be deployed [16].

The construction of a system with Fractal component yields a dynamically adaptable system where the component is the unit of configuration, deployment and reconfiguration. The system architecture, written in ADL, is expressed in terms of the component model, exhibiting bindings between components and containment relationships. These properties are specific to Fractal, compared to other component models, as explained more in detail in [4]. For these reasons, we chose to build our new AS using Fractal as well as the management system itself.

4. jonasALaCarte application server

jonasALaCarte is an AS prototype obtained by re-engineering JOnAS using the Fractal component model. Services and management entities have been wrapped into Fractal components. A jonasALaCarte instance is a composite component encapsulating a set of interacting services. The latter are bound using Fractal bindings. The administrator can choose the architecture of the AS, that is: how services use each other and in which JVM and where to be deployed. Management domains are implemented by Fractal composites that encapsulate other Fractal component under the same management policy.

4.1. jonasALaCarte architecture

Figure 4 presents a simple architecture of a jonasALaCarte. For the figure clarity, only part of the services is represented. The AS is a Fractal composite that contains two kinds of components: the services and the management entities.

![Figure 4. A Fractal-based architecture of jonasALaCarte](image)

Services in jonasALaCarte are Fractal components as well as the JOnAS management entities: the Configuration manager and the Loader manager. The Configuration manager represents a repository of default configurations. A service can be configured either by binding it to the Configuration Manager component or by setting its configuration parameters through the Attribute Controller. It is possible to reconfigure a service either by modifying the attribute controller parameters or by rebinding the service to a new Configuration manager. Each service is bound to
a Loader Manager to get its implementation configuration or classloader. A Scheduler component is introduced to start the services in an ordered way. For example, it is necessary to start the Security service before starting the Web service.

Relations between services and the relations between the services and the management components are expressed as Fractal bindings, represented by the arrows in the figure. For example, the EJB service is explicitly bound to the Transaction service it uses. The Web service and the EJB service are bound to the Configuration Manager component. The architecture of jonasALaCarte is specified thanks to an “Architecture Description Language” (jonasALaCarte.fractal). The instantiation of this ADL allows to configure and deploy the AS components.

4.2. Service re-engineering

In this section, we describe the modifications we introduced in the JOnAS service implementation. In JOnAS, a service extends an AbsServiceImpl class that allows mainly to start and stop the service by the ServiceManager class. A service accesses to an other service X through a static call to the Service Manager entity: ServiceManager.getInstance().getService(X). Calls to the configuration and the loader manager entities are also static. Behind these implicit relations, there is the assumption that all services and management entities run in a same JVM. In jonasALaCarte, a service has to extend an AbsServiceWrapper class. This abstract class implements mainly four kinds of interfaces: the Service interface to start or stop the service, the LifeCycleController that offers to a service an independent life cycle following JSR77, the ServiceConfController for service configuration and the BindingController to bind the service to the management components and to the other services through explicit Fractal bindings. The wrapper launches the “old” JOnAS service class using a classloader returned by the loader manager. The “old” service implementation is modified so that static calls are replaced by calls on the references transferred by the wrapper. The figure 5 illustrates the transformation we did in the service implementation.

The service ADL gives the necessary service information: the content tag represents the service wrapper implementation class, the attribute controller
that allows mainly updating the service implementation and the list of client and server interfaces that allow binding the service to the other AS components.

4.3. jonasALaCarte in a cluster environment

In this section, we illustrate the distribution aspect of jonasALaCarte. Similarly to the centralized case, a jonasALaCarte instance is made of a set of interacting services. Bindings between services are done using the Fractal RMI connector. The deployment of a distributed instance of jonasALaCarte is transparent for the administrator: he only has to produce an ADL file similar to centralized files, except that it contains the specification of nodes on which the different services have to be deployed.

Unlike current JOnAS configuration, we replicate services and not the whole servers. In fact, the EJB containers and Web containers are generally the bottleneck and we need more replicas for these services than for the others (Registry service, Transaction service etc.). Consequently, in jonasALaCarte, the replication is more selective and the architecture definition and presentation is the same as in a local JVM.

Furthermore, like in a single JVM, the architecture of the system is explicit. For example, in figure 6, Web containers and EJB containers share in the figure the same CMI registry, which can be replicated or not. Only two Transaction Manager Services (or TM) are deployed and a single DB is shared between the EJB containers.

![Figure 6. jonasALaCarte in a cluster environment](image)

Management units can be centralized. For example in the figure 6, all the EJB services are bound to a single Configuration Manager which represents actually a common configuration repository. This avoids, as currently in JOnAS, to replicate the same configuration on each replica.

Furthermore, the administrator has the ability to group a set of services under a same management domain. In figure 7, we introduce two domains: the Web tier domain and the EJB tier domain. This is interesting to apply common management policies. For example the Web domain can be under the control of a manager in charge of deploying the applications in a transactional way on all the Web services.

![Figure 7. Management entities and domains as Fractal components](image)

The managers are also Fractal components. Changing a management policy implies rebinding the domains to a new Manager component.

4.3. jonasALaCarte management

In this section, we summarize the management features we obtained by adopting a component-based approach and how we improve consequently the management in JOnAS. Unlike in JOnAS, in jonasALaCarte, the relations between services and the management components are explicit. This allows the coexistence of many configurations and the flexibility to reconfigure the system at run-time.

Adopting the service as a unit of management and implementing it as a Fractal component allows fine-grained management features in the AS. The deployment and configuration of jonasALaCarte use the configuration and deployment features in Fractal. The system architecture is defined in Fractal ADL and is instantiated using either local or distributed services linked with Fractal bindings, and a set of domain components to apply some management policies.

Many adaptation functionalities are possible. To change a service version, a new component with the new version is deployed, and all the bindings with the old component are replaced by bindings with the new one. At run-time, the manager can reconfigure a service through the Attribute Controller or by rebinding it to a new Configuration Manager. It is also possible to modify the dependencies between services.
using Fractal binding features, like binding an EJB service to a new DB.

In the distributed environment, Fractal containment property allows to define different management domains and to define hierarchy between them. This is a key feature that allows the isolation and the aggregation of management operations. This feature is inexistent in JOnAS.

Additionally, the management system and the managed services are seen by the manager in a uniform way: all the entities are Fractal components. This allows simplifying the manager task, since the system complexity is simplified to Fractal components and management operations are simplified to the manipulation of Fractal components such as adding or deleting, binding or rebinding the components.

It is important to mention, that our reengineering work does not break the conformance to J2EE standards: JSR77 and JSR88 implemented in JOnAS. In fact, the MBean instrumentation implemented in the services is preserved and the JMX service exposes the system content following JSR77 information model. The notion of domain is defined in JSR77 and is compatible with our usage of this tool in jonasALaCarte. Furthermore, the wrapping of the EJB and Web containers does not break the JSR88 conformant deployment tools embedded in the services. We aim only at adding further functionalities. J2EE management standards cover only some parts of the management functionalities of the AS. In JSR77, JMX instrumentation and connectors allow monitoring the system and exposing the system resources in a well-defined information model. However, the standard does not address the adaptation functionality. JSR88 considers the deployment of J2EE applications and not the middleware. Furthermore, the two standards does not address the distribution aspects of J2EE AS. Therefore, with respect to JOnAS that implements these standards, we enhanced the adaptation features; we offered a common tool for the deployment and configuration of the middleware in a single JVM and in a distributed environment.

Furthermore, since application components in J2EE communicate using RMI, the re-engineering of the middleware in jonasALaCarte is completely transparent to the application. In particular, there is no performance overhead on applications. The only overhead happens at the ignition time, when the system is installed.

5. Related work

Regarding the other open source application servers, we offer a flexible architecture that uses the same model for the middleware and the management system itself. In JBoss [11], the architecture of the AS is completely based on JMX. JMX offers a relationship service to express relations between MBeans. However, this service is rarely used for its complexity. In JBoss, some tools are built on top of JMX to express the dependencies. In jonasALaCarte, the relation between services are explicit thanks to the built-in Fractal binding features. In Geronimo [10] of Apache, the MBean model is enhanced to the GBean model. A deployment plan allows expressing relations and dependencies between GBeans like we do in Fractal ADL. A proxy is introduced between two GBeans to allow the transparent substitution of one GBean. However, like in MBeans, there is no notion of containment of GBeans as it exists in Fractal. Consequently, it is not clear how a GBean or a MBean model can be exploited to represent a management domain for instance. In jonasALaCarte, thanks to Fractal composition, we represent the system at different levels of abstractions: from services to domains. Furthermore, the MBean and GBean model does not cover the distribution requirement. In fact, an MBean/GBean component has to be deployed in the same JVM as the JMX server. We are unaware of MBean/GBean solutions for the distribution of services in JBoss/Geronimo similar to what we do in jonasALaCarte.

The problem of exposing the relations between the AS services is also raised in [9]. A management system is built on top of the AS (JBoss) to express the dependencies between services. Our approach is to modify the middleware itself and adopt a modular architecture where dependencies are explicit. SmartFrog [17] is a deployment framework that is applicable to the ignition and deployment of AS in a cluster environment. The system proposes a component model that allows encapsulating the cluster parts (Apache, Tomcat, AS etc.) and expressing the dependencies between these parts. But again, contrarily to Fractal, the model does not support the composition property which disallows management aggregation and multiple levels of abstractions of the system. Furthermore, in jonasALaCarte, we have more fine-grained unit of management (the service) whereas in SmartFrog, the AS is seen as a black box.

6. Conclusion and perspectives

In this paper, we have presented a new approach to improve the management features in a J2EE AS:
adopting a component-based architecture for the middleware and for the management system. We selected Fractal model to represent the services in the middleware which allows having fine-grained management features. The same model is used to represent the management entities at different levels of abstractions thanks to the notion of domains. This allows aggregating and isolating different management policies. Thus, the management system and the managed services are represented in a uniform way to the manager which simplifies significantly the administrator tasks. We presented an implementation of the component-based AS called jonasALaCarte by re-engineering JOnAS. The server can be configured and deployed either in a single machine or distributed in different machines. The cluster environment is just a particular configuration of the AS and does not involve additional tools.

The current implementation of jonasALaCarte allows an on-demand deployment and configuration, local or distributed, of the middleware and the management domains. In the near future, we aim at developing some reconfiguration policies such as recovery from failures and versioning. Furthermore, we aim at implementing autonomic management features by plugging the managers as Fractal components. We are also interested in generalizing our approach and implementing the J2EE application components as Fractal components. This would allow enhancing the management of J2EE applications as well and proposing a general management system for the middleware and the components using the same tools.

Acknowledgement

We would like to thank Professor Jacques Mossière for his review and precious remarks on this paper. Our thanks to the JOnAS project team for their interest in this work, particularly Adriana Danes and François Exertier. We also thank Jakub Kornas for his help to improve the paper content and presentation.

References