The JOnAS Open Source J2EE Platform

François Exertier

Table of Contents

This document presents the JOnAS J2EE platform. After a short introduction of the J2EE concepts, and of the ObjectWeb Consortium, the JOnAS server architecture and features are presented.

Introduction

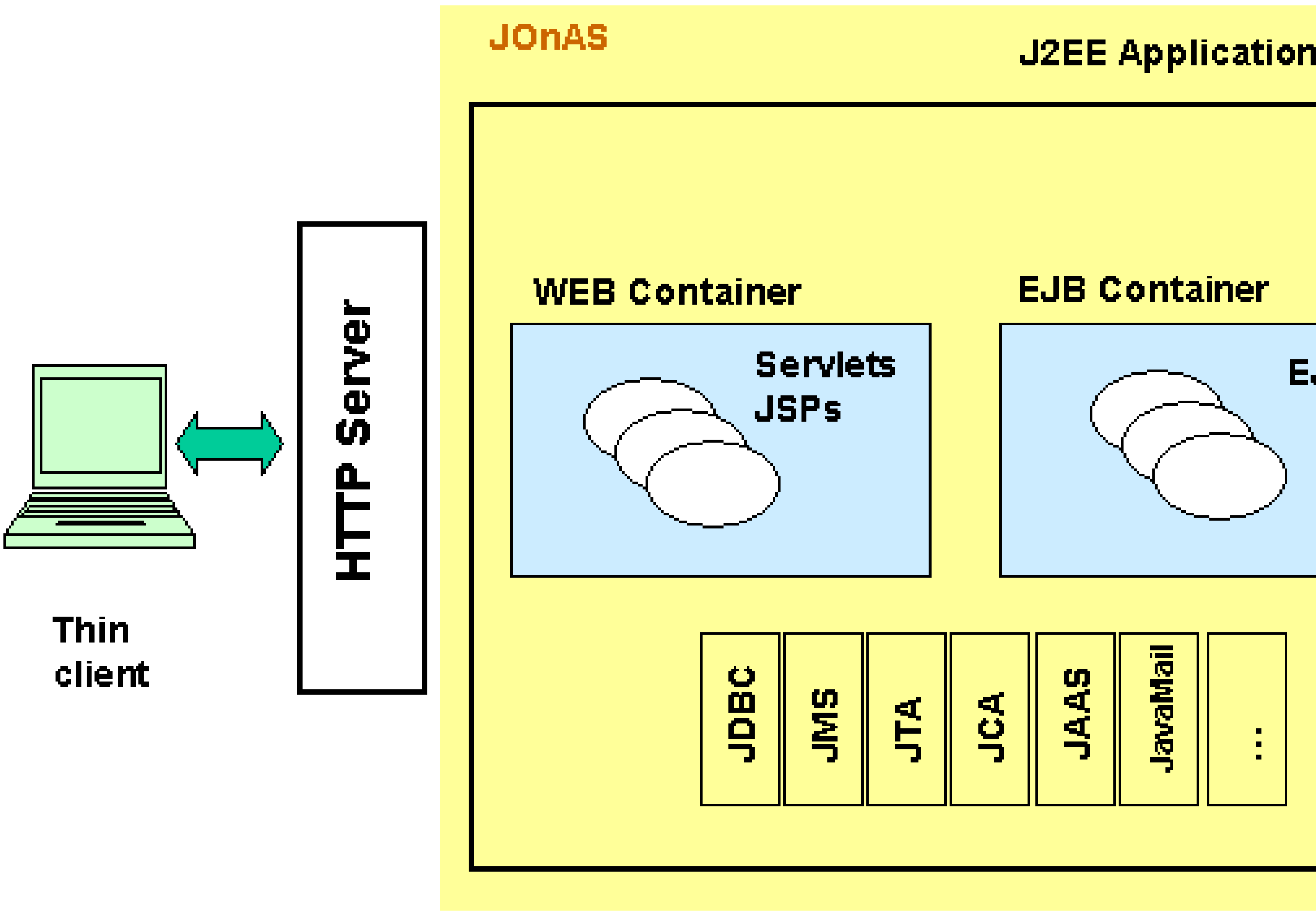
J2EE

The Sun J2EE specification [http://java.sun.com/j2ee/], together with its related specifications (EJB [http://java.sun.com/products/ejb/], JMS [http://java.sun.com/products/jms/],...), defines an architecture and interfaces for developing and deploying distributed internet Java server applications based on a multi-tier architecture. This specification intends to facilitate and normalize the development, deployment and assembling of applicative components; such components will be deployable on J2EE platforms. The resulting applications are typically web based, transactional, database-oriented, multi-user, secured, scalable and portable. More precisely, this specification describes two kinds of information:

- The first one is the runtime environment, called J2EE server, which provides the execution environment and the required system services such as the transaction service, the persistence service, the JMS service, the security service ...
- The second one is some kind of programmer's guide and user's guide explaining how an application component should be developed, deployed and used.

Not only will an application component be independent of the platform and operating system (since written in Java), but also of the J2EE platform.

A typical J2EE application is composed of presentation components, also called "web components" (Servlets [http://java.sun.com/products/servlet] and JSPs [http://java.sun.com/products/jsp]), which define the application Web interface, and of enterprise components, the "Enterprise JavaBeans", which define the application business logic and application data. The J2EE server provides containers for hosting web and enterprise components. The container provides the component life cycle management and interfaces the components with the services provided by the J2EE server. There are two kinds of container, the web container handles Servlet and JSP components while the EJB container handles the Enterprise JavaBeans components.



ObjectWeb

JOnAS is an open source implementation of J2EE, developed within the ObjectWeb consortium. ObjectWeb [http://www.objectweb.org] is an open source initiative which may be compared to Apache or Linux, but in the area of *middleware*. The aim of ObjectWeb is to develop and promote open source middleware softwares.

ObjectWeb is an International Consortium hosted by INRIA, officially founded in February 2002 by Bull, France Telecom, and INRIA. All softwares are available with the LGPL licence.

The technical objective of this consortium is to develop a distributed component-based middleware technology, in line with CORBA, Java and W3C standards. The idea is to apply the component model, as already used at the application level in J2EE or in the CORBA Component Model, at the middleware level by itself. The functional coverage of ObjectWeb projects addresses naming, trading, communication (events, messages), availability and safety (transactions, persistence, replication, fault tolerance), load balancing, and security. Some transversal topics are also addressed, like robustness, optimisation, code quality, as well as benchmarks, tests, evaluations, demonstrators and development tools.

Thus, the global ObjectWeb architectural model goes top down, from applications (as benchmarks such as Rubis from the JMOB project) running on middleware platforms (such as JOnAS, OpenCCM or ProActive). The platforms are based on technical components such as a Message-Oriented Middleware (JORAM which implements JMS), a communication framework (CAROL), a persistence framework (JORM), a transactional monitor (JOTM), a Object Request Broker (Jonathan). A technical component such as C-JDBC allows any platform to benefit from database clusters. JOnAS makes use of all these ObjectWeb components (JORAM, CAROL, Jonathan, JORM, C-JDBC, and soon JOTM), but is also using open source components from other communities, this is the case for Tomcat or Jetty, used as Web container, or for AXIS, used for providing the "Web Services" environment.

ObjectWeb has already a significant number of corporate members, universities, and also individual members (individual membership is free). ObjectWeb members contribute to ObjectWeb orientations, and participate to all ObjectWeb working groups, meetings, workshops and conferences. The community of developers and users around ObjectWeb components and platforms is growing continuously.

JOnAS Features

JOnAS is a pure Java open source application server conforming to the J2EE specification. However its high modularity allows to use it

- as a J2EE server, for deploying and running EAR applications (i.e. applications composed of both web and ejb components),
- as an EJB container, for deploying and running EJB components (e.g. for applications without web interfaces or when using JSP/Servlet engines that are not integrated as a JOnAS J2EE container),
- as a Web container, for deploying and running JSPs and Servlets (e.g. for applications without EJB components).

System Requirements

JOnAS is available for JDK 1.3 or 1.4. It has been used on many operating systems (Linux, AIX, Windows, Solaris, HP-UX, etc.), and with different Databases (Oracle, PostgreSQL, MySQL, SQL server, Access, DB2, Versant, Informix, Interbase, etc.).

Java Standard Conformance

JOnAS is an implementation of J2EE 1.3. It currently conforms to EJB 2.0. Its current integration of Tomcat or Jetty as Web container insures conformity to Servlet 2.3 and JSP 1.2 specifications. The JOnAS server relies on or implements the following Java APIs: JCA 1.0, JDBC 2.0, JTA 1.0.1, JMS 1.1, JMX 1.0, JNDI 1.2.1, JAAS 1.0, JavaMail 1.3.

Key Features

In addition to the implementation of all J2EE related standards, JOnAS provides the following advanced and important features:

- *Management:* JOnAS server management uses JMX and provides a servlet based managment console.
- *Services:* The service based architecture of JOnAS provides a high modularity and configurability of the server. It allows to apply a component model approach at the middleware level, and makes easy the integration of new modules (e.g. for open source contributors). It also allows to start only the services needed by a particular application, thus saving useless system resources. JOnAS services are manageable through JMX.
- *Scalability:* JOnAS integrates several optimization mechanisms for increasing the server scalability. This includes pool of stateless session beans, pool of message-driven beans, pool of threads, cache of entity beans, activation/passivation of entity beans, pool of connections (for JDBC and JMS), storage access optimizations (shared flag, isModified).
- *Clustering:* JOnAS clustering solutions, both at the WEB and EJB level provide load balancing, high availability and failover support.
- *Distribution:* JOnAS is working with several distributed processing environments, thanks to the integration of the CAROL [http://www.objectweb.org/carol/index.html] (Common Architecture for RMI ObjectWeb Layer) ObjectWeb project, which allows simultaneous support of several communication protocols:
  - *RMI* using the Sun proprietary protocol JRMP,
  - *RMI* on IIOP,
  - *CMI*, the "Cluster aware" distribution protocol of JOnAS,
  - or *Jeremie*, the RMI personality of an Object Request Broker called Jonathan [http://www.objectweb.org/jonathan/index.html from Objectweb. Used with Jeremie, JOnAS benefits from transparent local RMI calls optimization.
- *Support of "Web Services":* by the integration of AXIS, JOnAS allows J2EE components to access "Web services" (i.e. to be "Web Services" clients), and allows J2EE components to be deployed as "Web Services" endpoints.

Three critical J2EE aspects have been early implemented in the JOnAS server:

- *JCA:* Enterprise Information Systems (EIS) may be easily accessed from JOnAS applications. By supporting the Java Connector Architecture, JOnAS allows to deploy any JCA compliant Resource Adapter (connector) that will make the corresponding EIS available from the J2EE application components. For example, Bull GCOS mainframes may be accessed from JOnAS using their associated Hoox connectors [http://www.bull.com/servers/gcos8/products/interop8/hoox_pr.htm].
- *JMS:* JMS implementations may be easily plugged into JOnAS. They run as a JOnAS service in the same JVM or in a separate JVM and JOnAS provides administration facilities that hide the JMS proprietary administration APIs. Currently two JMS implementations may be used: the JORAM [http://www.objectweb.org/joram/] open source JMS implementation from Objectweb and SwiftMQ [http://www.swiftmq.com/].
- *JTA:* The JOnAS platform supports distributed transactions that involve multiple components and transactional resources. The JTA transactions support is provided by a Transaction Monitor that has been developed on top of an implementation of the CORBA Transaction Service (OTS).

JOnAS Packages

JOnAS is available for download with three different packagings:

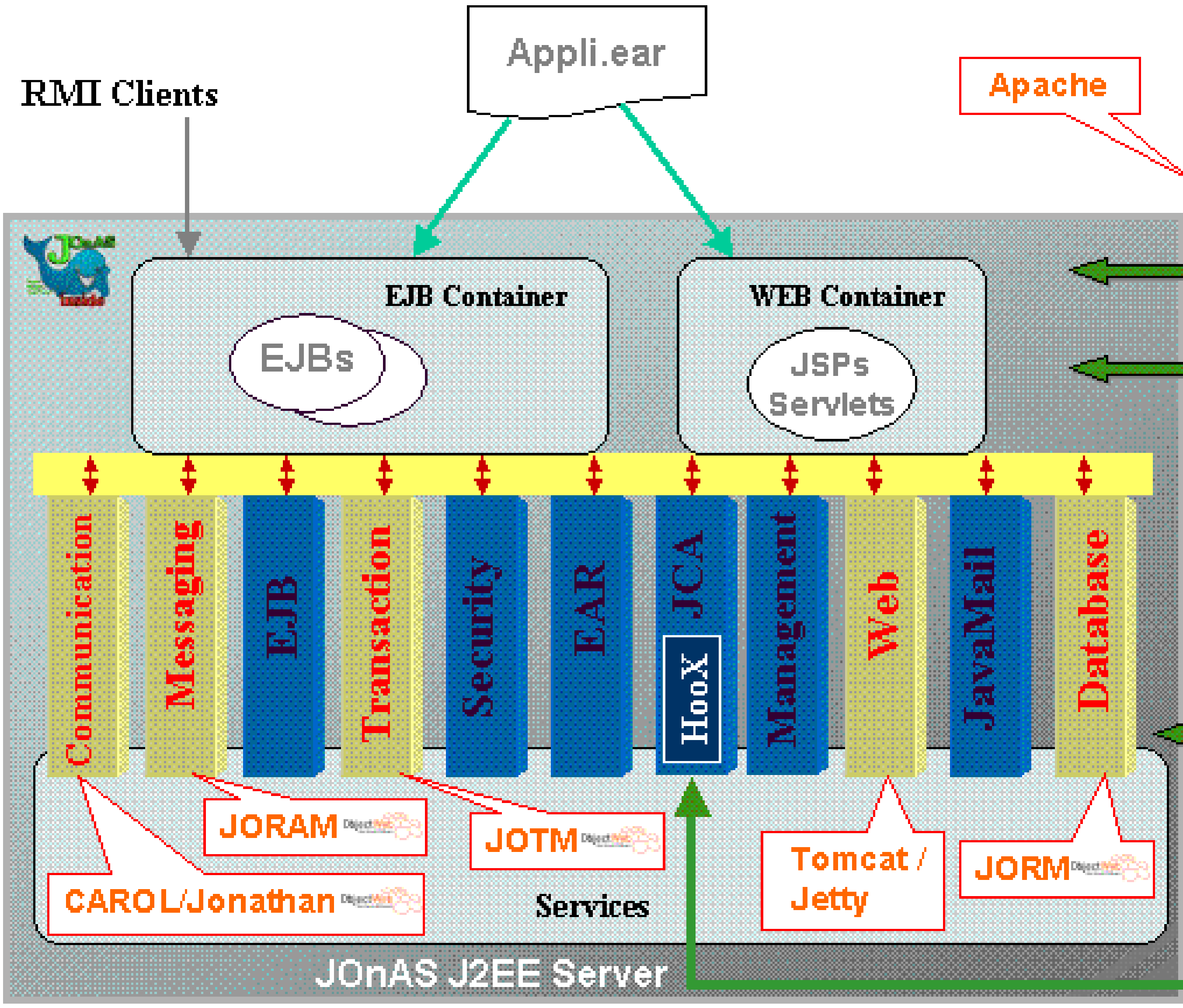
- a simple packaging only contains the JOnAS application server, without the associated Web Container implementation. In order to use it for J2EE Web applications, you need to have Tomcat or Jetty installed (with an adequate version) and to configure it and JOnAS so that they can work together.

- a JOnAS-Tomcat package contains both JOnAS and Tomcat, pre-configured, and with compiled examples. This is a ready to run J2EE application server. This package also contains AXIS, thus providing pre-configured "Web Services" support.
- a JOnAS-Jetty package contains both JOnAS and Jetty, pre-configured, and with compiled examples. This is a ready to run J2EE application server. This package also contains AXIS, thus providing pre-configured "Web Services" support.

JOnAS Architecture

JOnAS is architectured in term of services. A service typically provides system resources to containers. Most of the components of the JOnAS application server are pre-defined JOnAS services. However it is possible and easy for an advanced JOnAS user to define its own service and to integrate it into JOnAS. J2EE applications do not necessarily need all services, it is thus possible to define, at JOnAS server configuration time, the set of services that are to be launched at server start.

The JOnAS architecture is illustrated in the figure below, showing WEB and EJB containers relying on JOnAS services (all services are present on this figure). Two thin clients are also shown on this picture, one of them being the JOnAS administration console (called Jadmin).



Communication and Naming Service

This service (also called "Registry") is used for launching the RMI registry, the CosNaming, the CMI registry, and/or the Jeremie registry depending on the configuration of JOnAS (CAROL configuration which specifies which communication protocols are to be used). There are different registry launching modes, i.e. in the same JVM or not, automatically if not already running.

This service provides the JNDI API to application components and to other services, in order to bind and lookup remote objects (e.g. EJB Homes) and resource references (JDBC DataSource, Mail and JMS connection factories, etc.).

EJB Container Service

This service is in charge of loading the EJB components and their containers. EJB containers are composed of a set of Java classes implementing the EJB specification and by a set of interposition classes interfacing the EJB components with the services provided by the JOnAS application server. Interposition classes are specific to each EJB component and are generated by the deployment tool called GenIC.

JOnAS configuration allows to specify this service to be launched with a set of ejb-jar files to be loaded. Ejb-jar files may also be deployed at server runtime using the JOnAS administration tools.

Enterprise JavaBeans (EJB) are software components implementing the business logic of an application (while the Servlets and JSPs implement the presentation). There are three kinds of Enterprise JavaBeans:

- *Session beans* are objects associated to only one client, short-lived (one method call or a client session), that represent the current state of the client session. They can be transaction-aware, stateful or stateless.
- *Entity beans* are objects that represent data in a database. They may be shared by several clients and are identified by means of a primary key. The EJB container is responsible for managing the persistence of such objects. The persistence management of such an object is entirely transparent to the client that will use it, and may be or may not be transparent to the bean provider that develops it depending if it is

  - An enterprise bean with *Container-Managed Persistence*. In such a case, the bean provider does not develop any data access code, persistence management is delegated to the container. The mapping between the bean and the persistent storage is provided in the deployment descriptor, in an application server specific way.
  - An enterprise bean with *Bean-Managed Persistence*. In this case, the bean provider writes the database access operations in the methods of the enterprise bean that are specified for data creation, load, store, retrieval, and remove operations.

- *Message-driven Beans* are objects that may be considered as message listeners. They execute on receipt of a JMS (Java Message Service [http://java.sun.com/products/jms]) message, they are transaction aware and stateless. They implement some kind of asynchronous EJB method invocation.

For implementing Container-Managed Persistence of EJB 2.0 (CMP2), JOnAS relies on the ObjectWeb JORM [http://www.objectweb.org/jorm] (Java Object Repository Mapping) framework. JORM supports complex mappings of EJBs to database tables, and several kinds of persistency support (relational databases, object databases, LDAP repositories, etc.).

WEB Container Service

This service is in charge of running a Servlet/JSP Engine in the JVM of the JOnAS server, and of loading web applications ("war" files) within this engine. Currently this service may be configured to use Tomcat [http://jakarta.apache.org/tomcat/] or Jetty [http://www.jettyserver.org]. Servlet/JSP engines are integrated within JOnAS as "web containers", i.e. such containers provide the web components with access to the system resources (of the application server) and to EJB components, in a J2EE compliant way.

JOnAS configuration allows to specify this service to be launched with a set of war files to be loaded. War files may also be deployed at server runtime using the JOnAS administration tools. Tomcat and JOnAS users management has been unified. The class loading delegation policy (priority to the Webapp classloader or to the parent classloader) may be configured.

Servlet [http://java.sun.com/products/servlet] and JSP [http://java.sun.com/products/jsp] are technologies for developing dynamic web pages. The Servlet approach allows the development of Java classes (HTTP Servlets) that may be invoked through HTTP requests and that generate HTML pages. Typically Servlets access the information system using Java APIs (as JDBC or the APIs of EJB components) in order to build the content of the HTML page they will generate in response to the HTTP request. The JSP technology is a complement of the Servlet technology. A JSP is an HTML page containing Java code within particular XML-like tags; this Java code is in charge of generating the dynamic content of the HTML page.

Servlets and JSPs are considered as J2EE application components responsible of the application presentation logic. Such application components are able to access resources provided by the J2EE server (as JDBC datasources, JMS connection factories, EJBs, mail factories, ...). As for EJB components, the actual assignment of these resources is performed at component deployment time, and is specified in the deployment descriptor of each component (the component code use logical resources names).

Ear Service

This service is used for deploying complete J2EE applications, i.e. applications packaged in EAR files (themselves containing ejb-jar files and/or war files). This service will handle the EAR files, and will delegate the deployment of the war files, resp. ejb-jar files, to the WEB Container service, resp. to the EJB Container service. It will take care of creating the appropriate class loaders (as preconised in the J2EE specification) in order for the J2EE application to execute properly.

For deploying J2EE applications, JOnAS must be configured to launch the EAR service, and the set of EAR files to be loaded may be specified. EAR files may also be deployed at server runtime using the JOnAS administration tools.

Transaction Service

This is a Java Transaction Monitor called JOnAS JTM. It is a mandatory service which handles distributed transactions. It provides transaction management for EJB components as defined in their deployment descriptors. It handles two-phase commit protocol against any number of Resource Managers (XA Resources). For J2EE, a transactional resource may be a JDBC connection, a JMS session or a JCA Resource Adapter connection. The transactional context is implicitly propagated with the distributed requests. The JTM may be distributed across one or more JOnAS servers; thus a transaction may involve several components located on different JOnAS servers. This service implements the JTA 1.0 specification, thus allowing to explicitly start and terminate transactions from application components or from application clients. Starting transactions from application components is only allowed from Web components or from session beans or message-driven beans (and only these two kinds of beans, this is called *"Bean-managed transaction demarcation"*).

One of the main advantages of the EJB support for transactions is its declarative aspect, which means that transaction control is no longer hard coded in the server application, but is configured at deployment time. This is known as "*Container-managed transaction demarcation*".With "*Container-managed transaction demarcation*" the transactional behaviour of an enterprise bean is defined at configuration time, and is part of the deployment descriptor of the bean. The EJB container is responsible for providing the transaction demarcation for the enterprise beans, according to the value of transactional attributes associated to EJB methods which can be one of the following:

- *NotSupported:* if the method is called within a transaction, this transaction is suspended during the time of the method execution.
- *Required*: if the method is called within a transaction, the method is executed in the scope of this transaction, else, a new transaction is started for the execution of the method, and committed before the method result is sent to the caller.
- *RequiresNew:* the method will always be executed within the scope of a new transaction. The new transaction is started for the execution of the method, and committed before the method result is sent to the caller. If the method is called within a transaction, this transaction is suspended before the new one is started, and resumed when the new transaction has completed.
- *Mandatory*: the method should always be called within the scope of a transaction, else the container will throw the *TransactionRequired* exception.

- *Supports:* the method is invoked within the caller transaction scope, if the caller does not have an associated transaction, the method is invoked without a transaction scope.
- *Never*: with this attribute the client is required to call the method without a transaction context else the Container throws the java.rmi.RemoteException exception.

The JOTM [http://www.objectweb.org/jotm] (Java Open Transaction Manager) project from ObjectWeb has been created based on the JOnAS transaction service. It has been enhanced to provide advanced transaction features, such as nested transactions, "Web Services" transactions (an implementation of DBTP is available). JOTM will be integrated in JOnAS in a near future.

Database Service

This service is responsible for handling Datasource objects. A Datasource is a standard JDBC administrative object for handling connections to a database. The Database service creates and loads such datasources on the JOnAS server. Datasources to be created and deployed may be specified at JOnAS configuration time, or they may be created and deployed at server runtime using the JOnAS administration tools. The Database service is also responsible for connection pooling, i.e. it manages a pool of database connections to be used by the application components, thus avoiding many physical connection creations which are time consuming operations.

Security Service

This service implements the authorization mechanisms for accessing EJB components, as specified in the EJB specification. EJB security is based on the notion of *roles*. The methods can be accessed by a given set of roles. In order to access the methods, the user *must* be at least in one of this set of roles. The mapping between roles and methods (permissions) is done in the deployment descriptor using the security-role and method-permission elements. Programmatic security management is also possible using two methods of the EJBContext interface, in order to enforce or complement security check in the bean code: getCallerPrincipal() and isCallerInRole (String roleName). The role names used in the EJB code (in the isCallerInRole method) are in fact references to actual security roles, which makes the EJB code independent of the security configuration described in the deployment descriptor. The programmer makes these role references available to the bean deployer or application assembler by the way of the security-role-ref elements included in the session or entity elements of the deployment descriptor.

In JOnAS, the mapping between roles and user identification is done in the user identification repository. When using Tomcat for user authentification, this user identification repository may either be stored in files, in a JNDI repository (such as LDAP), or in a relational database. This is achieved through a JOnAS implementation of the Tomcat Realms, i.e. UserDatabaseRealm, MemoryRealm , JDBCRealm, JNDIRealm. These realms are in charge to propagate the security context to the EJB container during EJB calls. For Java clients authentification, JAAS login modules are provided.

Messaging Service

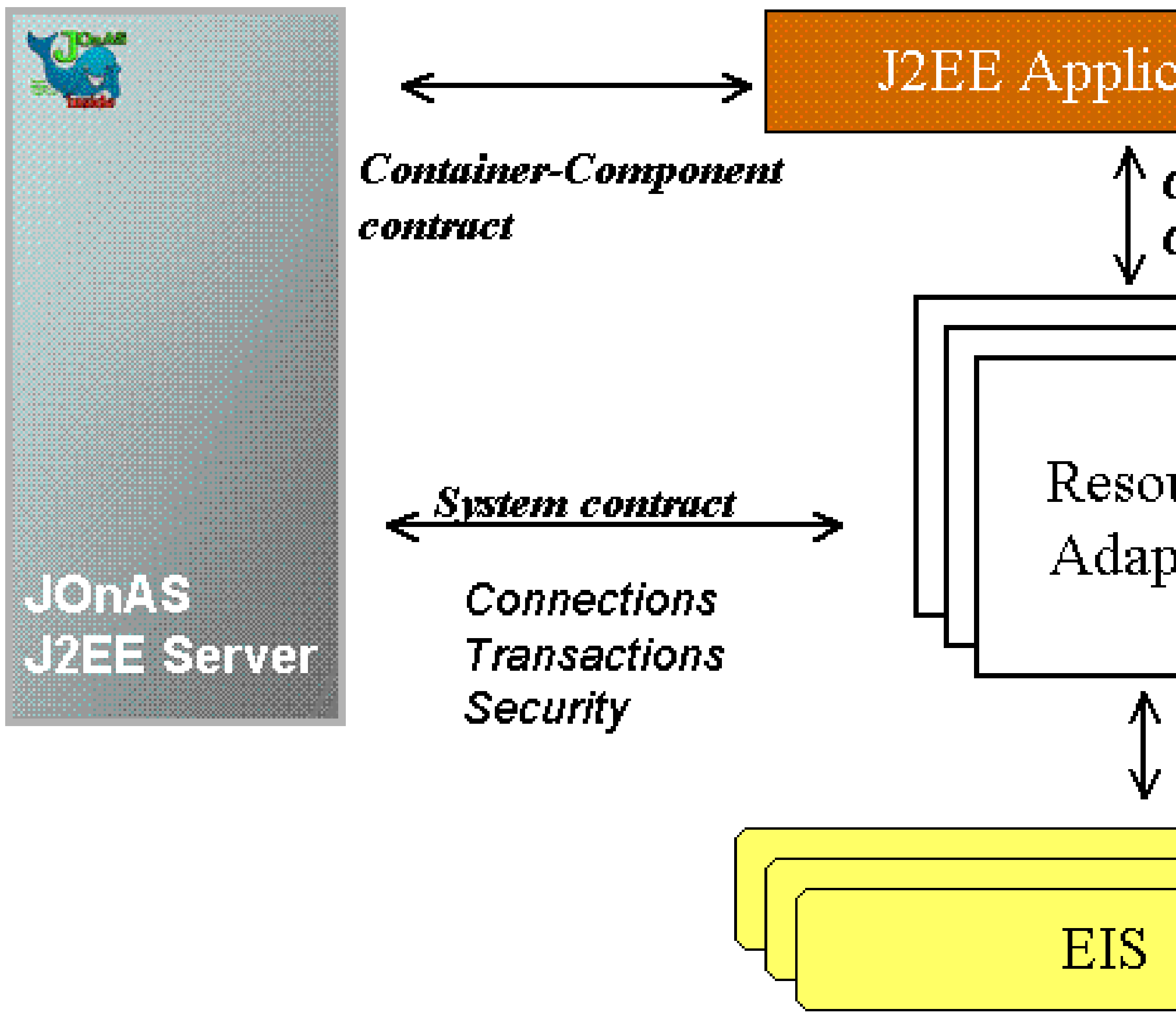
Asynchronous EJB method invocation is possible on Message-driven Beans components, as specified in the EJB 2.0 specification. A Message-driven Bean is an EJB component which may be considered as a JMS (Java Message Service [http://java.sun.com/products/jms]) MessageListener, i.e. which processes JMS messages asynchronously, it is associated with a JMS destination, its *onMessage* method will be activated on the reception of messages sent by a client application to this destination. It is also possible for any EJB component to use the JMS API, and this within the scope of transactions managed by the application server.

For supporting Message-driven Beans, and JMS operations coded within application components, the JOnAS application server relies on a JMS implementation. JOnAS makes use of a third party JMS implementation: currently the Joram [http://www.objectweb.org/joram] opensource software is integrated and delivered with JOnAS, the SwiftMQ [http://www.swiftmq.com/] product may also be used, other JMS provider implementations may easily be integrated. JORAM provides several points of interest, in particular reliability (with a persistent mode), distribution (it may run as several servers, transparently to the JMS client, thus allowing load balancing), the choice of TCP or SOAP as communication protocol for transmitting messages.

The JMS service is in charge of launching (or establishing connection to) the integrated JMS server, that may run in the same JVM as JOnAS or not. It also provides connection pooling, and thread pooling (for Message-driven Beans). Through this service, JOnAS provides facilities to create JMS administered objects such as the connection factories and the destinations either at server launching time, or at runtime using the JOnAS administration tools.

J2EE CA Resource Service

The J2EE Connector Architecture (J2EE CA) allows the connection of different Enterprise Information Systems (EIS) to a J2EE application server. It is based on the Resource Adapter (RA), an architecture component comparable to a software driver, connecting the EIS, the application server, and the enterprise application (J2EE components). The RA is generally provided by an EIS vendor, and provides a Java interface (the Common Client Interface or CCI) to the J2EE components for accessing the EIS (this may also be a specific Java interface). The RA also provides standard interfaces for being plugged to the application server, so that they can collaborate to keep all system-level mechanisms (transactions, security, and connection management) transparent from the application components.



The application performs "business logic" operations on the EIS data using the RA client API (CCI), while transactions, connections (including pooling) and security on the EIS is managed by the application server through the RA (system contract).

The JOnAS Resource service is in charge of deploying J2EE CA compliant Resource Adapters (connectors), packaged as RAR files, on the JOnAS server. RAR files may also be included in EAR files, in this case the connector will be loaded by the application classloader. Once Resource Adapters are deployed, a connection factory instance is available in the JNDI namespace to be looked up by application components.

Management Service

The Management service is needed in order to administrate a JOnAS server from the JOnAS Jadmin administration console. Each server running this service is visible from the administration console. This service is based on JMX. Standard MBeans are defined within the JOnAS application server, they expose the management methods of the instrumented JOnAS server objects such as services, containers, the server by itself. The Management service runs a JMX server (currently the MX4J one, the Sun RI one is also available). The MBeans of the JOnAS server are registered within this JMX server. The JOnAS administration console is a Web application (servlet/JSP based), which accesses the JMX server in order to present the managed features within the administration console. It is thus possible, through a simple Web browser, to manage one or several JOnAS application servers. The administration console allows to configure all JOnAS services (and to make the configuration persistent), to deploy any kind of applications (EJB-JAR, WAR, EAR) and any kind of resources (DataSources, JMS and Mail connection factories, J2EE CA connectors), all that without the need to stop/restart the server. The administration console shows many information allowing to monitor your servers and applications: used memory, used threads, number of EJB instances, which component currently uses which resources, etc..

Mail Service

A J2EE application component may send e-mails using JavaMail [http://java.sun.com/products/javamail/]. The Mail service of the JOnAS application server provides the necessary resources to such application components. The same way the database service or the JMS service create Datasources or ConnectionFactories and register these objects in the JNDI namespace, the Mail service creates mail factories and register these resources in the JNDI namespace. There are two kinds of mail factories: *javax.mail.Session* and *javax.mail.internet.MimePartDataSource*.

JOnAS Development and Deployment Environment

JOnAS Configuration and Deployment Facilities

Once JOnAS has been installed, in a directory referred by the JONAS_ROOT environment variable, it is possible to configure servers and to deploy applications into several execution environments. This is achieved using the JONAS_BASE environment variable. JONAS_ROOT and JONAS_BASE may be compared to CATALINA_HOME and CATALINA_BASE variables of Tomcat. While JONAS_ROOT is dedicated to JOnAS installation, JONAS_BASE is used to specify a particular JOnAS instance configuration. JONAS_BASE designates a directory containing a specific JOnAS configuration, and subdirectories containing the EJB-JAR, WAR, EAR, and RAR files that may be loaded in this application environment. There is an ANT target in the JOnAS build.xml file to create a new JONAS_BASE directory structure. So from one JOnAS installation, but just changing the value of the JONAS_BASE variable, it is possible to switch from one application environment to another. For configuring a JOnAS application server and for loading applications, you should use the administration console, or edit the configuration files. There are also "autoload" directories for each kind of application (EJB-JAR, WAR or EAR) that allow the JOnAS server to automatically load the applications located in these directories when starting.

About deployment, JOnAS provides several facilities.

- First to write the deployment descriptors, plugins for Integrated Development Environments (IDE) provide some generation and editing features (Eclipse and JBuilder plugins are available). The NewBean JOnAS built-in tool generates template deployment descriptors. The Xdoclet tool also generates deployment descriptors for JOnAS. The Apollon [http://debian-sf.objectweb.org/projects/apollon] ObjectWeb project generates Graphical User Interfaces for editing any XML file, it has been used to generate deployment descriptor editor GUI. A tool developed by the ObjectWeb JOnAS community will also be available to work with the JSR88 compliant (J2EE 1.4) deployment APIs provided by the ObjectWeb Ishmael [http://debian-sf.objectweb.org/projects/ishmael] project.

- For the deployment by itself, the basic tools are the JOnAS GenIC command line tool and the corresponding ANT ejbjar task. The IDE plugins integrate the use of these tools for deployment operations. The Ishmael project main feature will be the deployment of applications on the JOnAS platform.

JOnAS Development Environments

There are many plugins and tools for making easier the development of J2EE applications to be deployed on JOnAS. IDE plugins for JBuilder and Eclipse allows to develop, deploy and debug J2EE components on JOnAS. The Xdoclet [http://xdoclet.sourceforge.net/] code generation engine is able to generate EJB interfaces, deployment descriptors (standard and JOnAS specific ones), taking as input the EJB implementation class containing specific JavaDoc tags. The JOnAS NewBean tool generates templates of interfaces, implementation class, and deployment descriptors for any kind of EJB. Many development tools may work with JOnAS, see the JOnAS tools page [http://www.objectweb.org/jonas/tools.html] for more details.

Moreover, JOnAS is delivered with complete J2EE examples, providing build.xml ANT file with all necessary targets for compiling, deploying and installing J2EE applications.

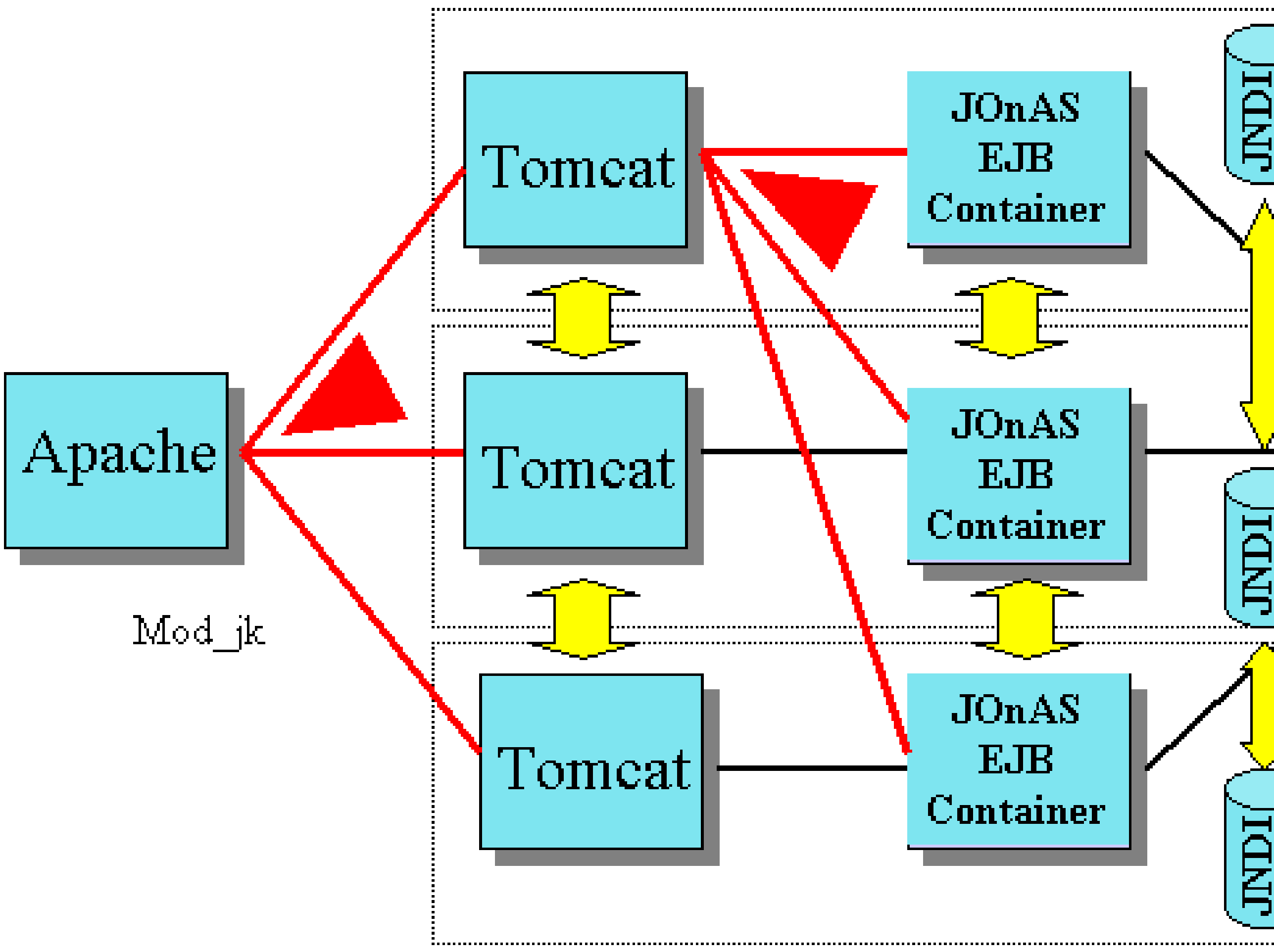
Clustering and Performance

Clustering for an application server generally brings three features: Load Balancing (LB), High Availability (HA) and Failover. Such mechanisms may be provided at the Web container level (by dispatching requests to several Servlet/JSP engine instances, at the EJB container level (by dispatching EJB requests to several EJB container instances), and at the database level by using several databases. A replicated JNDI naming is also necessary.

JOnAS provides Load Balancing, HA and Failover at the WEB container level, using the Apache Tomcat mod_jk plugin and an HTTP in memory session replication mechanism based on JavaGroup. The plugin dispatches HTTP requests from the Apache web server towards Tomcat instances running as JOnAS web container. Servers going down and up are automatically taken into account. This plugin supports round-robin and weighted round-robin load balancing algorithms, with a sticky session option.

Load balancing and HA is provided at the EJB container level in JOnAS. Operations invoked one EJB Home interfaces (EJB creation and retrieving) are dispatched on the nodes of the cluster. The mechanism is based on a "clustered aware" replicated JNDI registry using a Clustered remote Method Invocation protocol (CMI). The stubs contain the knowledge of the cluster, and implement the load balancing policy, which may be round-robin, and weighted round-robin. In a near future, a load balancing mechanism based on the nodes load will be available. Failover at the EJB level will be provided by implementing a stateful session bean state replication mechanism.

The JOnAS clustering architecture is illustrated in the figure below.



Apache is used as the font-end HTTP server, Tomcat as the JOnAS web container. The JOnAS servers share the same database. The mod_jk plug-in provides Load Balancing / High Availability at the Servlet/JSP level. Failover is provided through the in-memory session replication mechanism. Load Balancing / High Availability is provided at the EJB level through the CMI protocol associated with the replicated clustered aware JNDI registry. Tomcat may run in the same JVM as the EJB container or not. JOnAS provides some documentation for configuring such an architecture.

The use of the C-JDBC [http://www.objectweb.org/c-jdbc/index.html] ObjectWeb project allows to provide load balancing and high availability at the database level. The use of C-JDBC is transparent to the application (in our case to JOnAS), as it is viewed as a usual JDBC driver. However this "driver" implements the cluster mechanisms (reads are load balanced and writes are broadcasted). The database is distributed and replicated among several

15

nodes and C-JDBC load balances the queries between these nodes. An evaluation of C-JDBC using the TPC-W benchmark on a 6 nodes cluster has shown performance scaling linearly up to 6 nodes.

In addition to clustering solutions, JOnAS provides many mechanisms, intrinsic to the JOnAS server, for being highly scalable and efficient. This includes

- Pool of stateless session bean instances
- Pool of entity bean instances, configurable for each entity bean, within its deployment descriptor
- Activation/passivation of entity beans, passivation may be controlled through the management console
- Pools of connections, for JDBC, JMS, J2EE CA connectors
- Pool of threads for message-driven beans
- Session bean timeout may be specified at deployment
- A "shared" flag in the specific deployment descriptor of an entity bean indicates if the persistent representation of this entity bean is shared by several servers/applications, or if it is dedicated to the JOnAS server where it is loaded. In the last case, the optimization done by JOnAS consists in not reloading the corresponding data between transactions.
- The usual EJB 1.1 "isModified" (or "Dirty") mechanism is available, avoiding to store non modified data.

Some benchmarks and JOnAS usecases have already shown that JOnAS was highly scalable, see for example the Rubis [http://www.cs.rice.edu/CS/Systems/DynaServer/perf_scalability_ejb.pdf] results, or the OpenUSS [http://openuss.sourceforge.net/openuss/] usecase. Rubis is a benchmark for e-commerce J2EE applications, which now belongs to the ObjectWeb JMOB [http://www.objectweb.org/jmob/index.html] (Java Middleware Open Benchmarking) project. OpenUSS is an operational University portal with about 5000 users.

Perspectives

As an open source implementation of a J2EE server, JOnAS is continuously evolving to satisfy the user requirements and to follow the related standards. The main JOnAS evolutions currently planned are the following:

- A J2EE CA 1.0 Resource Adapter for JDBC is planned for the middle of the year. It will replace the current JOnAS database service for plugging JDBC drivers and managing connection pools. In addition, it will provide JDBC PreparedStatement pooling. At the end of the year, a new version of this resource adapter, compliant to J2EE CA 1.5 (J2EE 1.4), will be available, and will provide JDBC resultsets re-use by the EJB container (this optimization reduces the number of SQL statements executed by the EJB container).
- The support of EJB 2.1 (J2EE 1.4) specification will be provided by the end of the year. This will include a "Timer" service.
- The support of "Web Services" will be completed, in order to support standard deployment, as specified in J2EE 1.4. Tools for developing "Web Services" and an open source UDDI implementation will be integrated.
- Deployment APIs as specified in JSR88 (J2EE 1.4) will be supported by the end of the year, thanks to the Ishmael project.
- JOnAS administration will be enhanced by introducing the concept of management domain.
- The ObjectWeb JOTM project will be integrated within the JOnAS Transaction service. This will allow to support nested transactions, CORBA transactions and "Web Services" transactions.