



NXJ Developer Interaction Server Architecture

Unify Corporation



Overview

An NXJ application is a standard J2EE application. It is packaged in a J2EE enterprise archive (an “ear” file) and consists of J2EE components, namely a web application (packaged in a “war” file) and Enterprise JavaBeans (packaged in an EJB “jar” file). The “ear” file can be deployed into any J2EE platform. The section “Creating an NXJ Application – the Make Process” below describes the files that make up these J2EE components and how they are produced from the project source.

Once deployed, an NXJ application takes advantage of the J2EE platform to deliver a rich, highly interactive application experience for the end user. The web application interacts with the browser to manage the presentation. The Enterprise JavaBeans enforce the business rules concerning the data, manage the set of selected rows for each data view, and performs the built-in interaction with data sources. The section “Running an NXJ Application” describes how this is accomplished.

Below are recommended system configurations for NXJ. (Note these are recommended, not the minimal requirements). These recommendations come from Unify’s Services Organization which has built and deployed several NXJ customer applications.

- 1> Desk top or Tower: (i.e. Dell PowerEdge SC1420)
 - a. Dual 3.xGhz Xeon Processor
 - b. 2GB RAM
 - c. 160GB RAID 1 SATA Hard Drive (3 drives)
 - d. Fast network Adaptor (1000MT)
 - e. CD/DVD ROM
 - f. Mouse/Keyboard/Monitor
 - g. Windows Server 2003

- 2> Rack Mount server (i.e. Dell PowerEdge SC1425 or 1850)
 - a. Dual 3.xGhz Xeon Processor
 - b. 2GB RAM
 - c. 160GB RAID 1 SATA Hard Drive (3 drives)
 - d. Fast network Adaptor (1000MT)
 - e. CD/DVD ROM
 - f. Mouse/Keyboard/Monitor
 - g. Windows Server 2003

Running an NXJ Application

The NXJ Application consists of two main components: a web application component and an Enterprise JavaBeans component. These two components interact with each other and with both the browser and data sources to deliver an integrated application. Both of these two main components are discussed in turn.

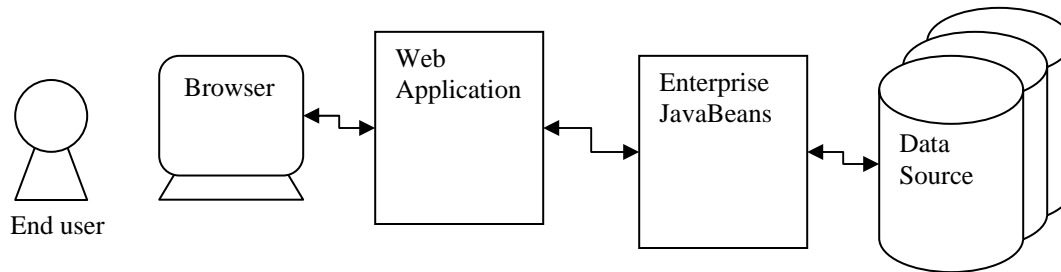


Diagram 1: Runtime Application Components

The Web Application

The web application is deployed into a Java J2EE Application Server that provides HTTP access to end users running browsers. The end user directs his browser to the published URL of the NXJ web application and the web app delivers to the browser the information necessary to present the application forms and commands to the end user. The web application uses a wide variety of web standards and protocols to deliver the Active Web behavior to the browser. These standards and protocols include HTML, JSP, JavaScript, GIF, JPEG, CSS, HTTP, HTTPS, SSL, and more. They do not include Java applets nor do they require installing any custom plug-in to the browser. NXJ applications run within standard, unmodified browsers.

The logic in the web application is tuned to minimize and manage the network traffic and to provide a smooth presentation to the end user. Commands issued by the user are evaluated on a form and field basis to determine whether a round trip to the

.....

server is necessary. If not, obviously the command is handled in the browser without additional network traffic. If the round trip is required then the communication is optimized to only pass data if possible. Returning an entire page to the browser is done only when it is necessary because the application has navigated to a different page in the application.

The pages of the application correspond to the forms created in the project. The concepts and controls available to the developer when laying out forms also lend themselves to optimizing the communication. Data view and tab sets allow the developer to build sophisticated pages with several multi-record data sets so that applications delivered to a browser can provide more functionality on a single page. The data sets can be related with built-in master-detail relationships that allow the user to interact with a complex, layered set of data on a single screen.

The Enterprise JavaBeans

The business logic for the application is separated from the web application and placed into an Enterprise JavaBean. This EJB is a stateful session bean that manages the state of the application. Separating the business logic from the web application provides a couple of advantages. One is that the application can be easily distributed across multiple machines. This provides more horsepower for supporting heavier application loads. It also allows the application to be distributed in environments where the network architecture enforces a separation of outward facing HTTP services and critical internal data resources. Even though the code is organized into an EJB the J2EE framework provides fast and efficient mechanisms, such as local EJB references and local invocation optimizations, for communicating to this EJB in environments where the web application and the EJB are co-located.

Another advantage of architecting the application as an EJB is that the business logic can take advantage of the more powerful transaction semantics of Enterprise JavaBeans. This allows NXJ to give the developer more flexibility in defining the boundaries of application transactions. In addition, application transactions in NXJ take full advantage of the J2EE framework. J2EE resource references are used to establish connections to the data sources used in the application. The transaction management capability of the J2EE application server is used to control these transactions. Therefore these transactions can include multiple operations across multiple data sources. This translates into operations on databases from different vendors, for example. The J2EE application server provides the distributed, two-phase commit behavior needed to ensure the consistency and atomicity of these transactions.

The code placed into form scripts is converted into a Java class that is a subclass of the NXJForm class in the NXJ framework. (The generated Java code can be found under the project's output directory.) Inner classes are generated for the data views, fields, and commands in the form. These generated form classes, along with the other Java classes that are part of the project, are packaged together into the stateful session EJB that manages the state of the application.

.....

Note that only a small portion of the application consists of generated Java code. The web application, for example, consists largely of JSP, XML, and HTTP files. A substantial portion of the behavior in both the web application and the EJB comes from the framework classes that were written in plain Java. It is the code in the framework classes that provides user interface coordination, implements the binding of controls to data elements, manages the selected set of records and query-by-forms capability in each data view, integrates with data sources, provides the built-in transaction management behavior, and integrates with the J2EE security framework, for example. The generated Java code provides the automatic declaration of variables for the fields, commands, and other elements in the layout as well as the common JDBC code produced from ANSI standard SQL statements embedded using the same style found in other programming languages.

Data Validation Example

This example illustrates how data validation is performed by the various component of the NXJ application. Suppose there is a field on a form that represents a promised delivery date. The end user types a value into this field and issues an Update command. The value typed by the user is validated in the browser to make sure that it is a well formed date value. If it is not a well formed date then an error message will be displayed and the user will know to correct the value. This all takes place without any communication with the server.

If the value entered by the user is a valid date value, then the Update command is sent from the browser to the web application. The Servlet in the web application forwards this request to the stateful session EJB that is managing the application. The EJB processes the edits to the fields, executing event sections such as ON DATA ACCEPT and WHEN VALUE CHANGES on the field if they exist. Since this is a promised delivery date, suppose we have a Web Service or stored procedure that will verify that the promised date is achievable. Further, if the date is refused, the service or procedure will return an alternative date that is achievable. The ON DATA ACCEPT event section for our promised date field, PROMISED_DATE, might look like this:

```
ON DATA ACCEPT
{
    java.sql.Date    okDate;
    okDate = verifyDeliveryDate(PROMISED_DATE.getDate());
    if ( okDate != null )
    {
        session.displayToMessageBox(
            "Requested date is not achievable. "
            + "Will this alternate date work?");
        PROMISED_DATE.setDate(okDate);
        rejectOperation();
    }
}
```

Notice that, if the date is not accepted, an error message is reported, the field is changed to the alternative date, and the data edit is rejected. Since the field edit was rejected, the Update operation will be automatically cancelled. The EJB responds back through the Servlet with the error message and a new, alternative value for the field. The user sees the error message and the date field has focus, showing the alternate date.

Creating an NXJ Application – the Make process

An NXJ application is built whenever one of the Make, Run, or Debug commands is invoked. The Make process involves processing and compiling source files to produce all the pieces of the J2EE application. The pieces are packaged into a J2EE enterprise archive, as illustrated in Diagram 2.

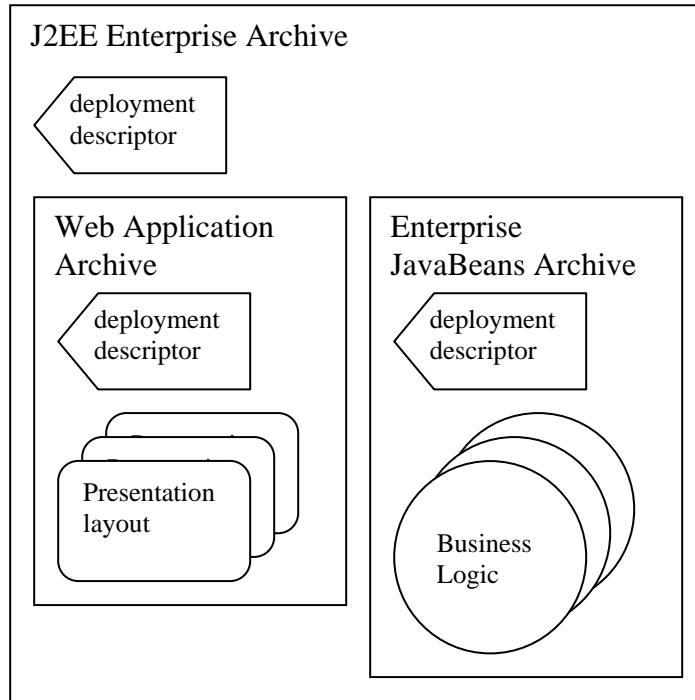


Diagram 2: J2EE Enterprise Archive Contents

About Unify

Unify is a global provider of software development technology and solutions that helps IT customers participate in Service-Oriented Architecture (SOA). Unify's productive and re-liable development tools, migration solutions and databases enable organizations to build and modernize business essential applications for SOA. Composer for Lotus Notes offers a complete, like for like, production to production migration solution for Lotus Notes applications. Unify's award-winning NXJ Developer enables IT teams to be extremely productive, learn new technologies fast and deliver Web services-based applications on time and on budget. The Team Developer, SQLBase, DataServer, ACCELL and VISION product families enable cross-platform rapid development on Java/J2EE, Linux or Windows. Unify has a rich heritage in delivering rich, cost-effective technologies to its thousands of IT customers and ISV, VAR and distributor partners. Unify is headquartered in Sacramento, Calif., and can be reached at (916) 928-6400 or by visiting www.unify.com.

Unify Corporation
2101 Arena Blvd., Suite 100
Sacramento, CA 95834
USA
Phone: 1.916.928.6400
Toll Free: 1.800.468.6439
Fax: 1.916.928.6404
Munich: +49 8 115 55430
United Kingdom: +44 (0)1753 245 510
France: +33 (0)1 34 58 28 30

COPYRIGHT © 2007. UNIFY CORPORATION. All rights reserved.

Unify, the Unify logo and Unify NXJ are registered trademarks of Unify Corporation.
Composer is a trademark of Unify Corporation.
Java and J2EE are the trademarks or registered trademarks of
Sun Microsystems, Inc. in the United States and other countries.
All other company or product names are trademarks of their respective owners.