Lotus Domino and IBM WebSphere integration for developers

Peter Weber

June 28, 2004

Many customers have integrated IBM Lotus Domino and IBM WebSphere into their environments. This article explains how you can build applications that take advantage of the relative strengths offered by both Domino and WebSphere.

Integration. For many, it's the buzzword of the new century, but what does it really mean and how can you use it to solve your application development needs? IBM WebSphere technology with its integrated solutions can solve business, organizational, and technological problems. But to do so, those involved in the architecting, planning, and implementation of the solution must agree upon the project's objectives and the approaches they will use to achieve them.

This article examines several issues involved in integrating WebSphere with Lotus Domino. We present a concise overview of how WebSphere/Domino integration can be realized to get the best out of these two different systems. This article assumes that you're an experienced Domino and/or WebSphere developer.

The J2EE standard

J2EE (Java 2 Enterprise Edition) is a specification for a standard platform for developing and hosting reusable components. The components can then be used to provide multi-tier client- and server-based enterprise applications. There are many implementations of J2EE. These include the original reference by Sun Microsystems, IBM's WebSphere Application Server, WebLogic by BEA, JRun by Macromedia, and the open source J2EE-compliant server JBoss. This article focuses on WebSphere, although it is also applicable to applications developed on any of the preceding J2EE-compliant platforms. (Our descriptions apply to both WebSphere v4 and v5. Although IBM WebSphere v4.0.x is J2EE 1.2 compliant, it already includes many J2EE 1.3 features.)

WebSphere’s approach is to separate all acting parts of an application into components. These components handle different topics of the complete application. For Domino developers, this separation is significantly different from what you do when designing and developing Domino applications. In Domino, for example, there's no separation of the UI and the business logic. The coding happens directly in forms, using actions or agents to perform the required tasks (though it's possible to separate the connection from the Domino application to an RDBMS or a legacy system with Lotus Enterprise Integrator, for instance). It's nearly impossible to have a
single Domino component responsible only for the UI and another dedicated solely to the business logic.

**J2EE components**

The WebSphere approach is completely different. Figure 1 shows how WebSphere separates the four components of a J2EE application (client, user component, server component, and legacy component).

**Figure 1. How WebSphere separates J2EE components**

The J2EE standard specification goes into more detail by taking a look at how the components interact:

**User component/server component interactions**

These include:

- Logging
- Instantiation
- Termination
- Interaction with back-end components:
  - Finding services
  - Registering services
  - Accessing services

**Client/server component interactions**

These include:

- **Protocols**
  These are for back-end access and include RMI/IIOP (Remote Method Invocation/Internet Inter-ORB Protocol) and RMI/JRMP or JDBC (Java Database Connectivity) with various driver types.

- **Transactions**
  Transactions should not be initiated from the client. It's recommended that transactions should be totally transparent for the client and therefore, used in the EJB (Enterprise JavaBean).

- **Security**

**Legacy component/server component interactions**
These include integration with external resources:

- Transactions
- Finding/registering/accessing services
- Java services
  - JDBC
  - JMS (Java Message Service)
  - JavaMail and more
- Connector architecture
  - EIS (Enterprise Information Systems)
  - JCA (Java Connection Architecture)

This is not supported in WebSphere Application Server Express.

When architecting a J2EE application, you need to know how to define the basis of displaying pages to the user, how to handle the input, how to implement the business logic, and how to connect to various data sources. This separation is based on the J2EE specification, so every J2EE-compliant system can be responsible for a fragment in the whole solution.

**J2EE architecture**

The J2EE architecture is based on a four-tier layer model that characterizes a J2EE-compliant enterprise application. These layers are the Application, Presentation, Business Logic, and Data layers.

- The Application layer contains the applications running in Application Client Containers, which are Java programs. The Application layer also includes applets, which are GUI components that execute in Web browsers.
- The Presentation layer contains servlets. These are similar to applets, but they run on the server rather than the client. This layer also contains JSPs. These execute on the server to provide responses.
- The Business Logic layer contains EJBs (Enterprise Java Beans). These execute on the server to manage business logic.
- The Data layer contains database systems (such as DB2 or Oracle) for holding data of any kind.

Together these four layers contain the components that represent the complete enterprise application. Each component of the application runs inside a container. The container itself is provided by a J2EE platform provider. A container supplies specific deployment and runtime services to the component (life cycle, security, transactions, and so on). Which services are offered depends on the container.

The Presentation layer and the Business Logic layer can be on two different machines and can be offered by different J2EE platform providers. For instance, the Presentation layer (Web container) can be provided by Tomcat and the Business Logic layer (EJB container) can run on WebSphere. This demonstrates that the solution is not concentrated on one single platform provider only; the specifications mandate that every J2EE-compliant system must support a Presentation layer and a Business Logic layer.
In modern J2EE architectures, the main focus is on scalability and high availability. Therefore, these four "classic" layers can be split into additional tiers, including client, Web, middleware, and others. And the concept of distribution can be expanded further. For instance, it's possible to have every EJB reside on a different machine. (In that case, you could improve performance by covering the EJB with a session bean with local access to reduce the number of remote accesses.) Also, data sources should be stored on a separate machine.

**Integrating Domino and WebSphere**

If you're an experienced Domino developer, you know there are many applications for which Domino is especially suited (for example, collaboration, handling unstructured data, and general presentation) and a few for which it is not (transactional processing and heavy business logic, for instance). WebSphere, on the other hand, excels at heavy transactional processing, distributed processing and connectors, scaling, and high availability. By integrating Domino and WebSphere, you can take advantage of each product's strengths and create applications that deliver more functionality than either platform can provide on its own.

A typical integrated Domino/WebSphere environment is shown in Figure 2.

**Figure 2. Domino/WebSphere integrated environment**

Domino/WebSphere integration consists primarily of browser integration, connecting Domino to WebSphere and connecting WebSphere to Domino. You will typically use all three together,
depending, of course, on the solution you need to deliver. We will discuss each of these in the following sections.

**Browser integration**

As illustrated in Figure 2, the browser client switches URLs from Domino to WebSphere. This happens via the server address or URI. This means the browser communicates either directly with Domino or directly with WebSphere. The information exchange with WebSphere can happen indirectly via a WebSphere plug-in (through Domino) or indirectly via an HTTP server. When the browser communicates with WebSphere, it does not matter whether it is talking to Domino or to another HTTP stack (unless you are using HTTPS). For routing requests to WebSphere, WebSphere supplies a plug-in for each HTTP stack.

The advantage of browser integration is the ability to use different development teams. On the downside, browser integration requires you to ensure users experience the same "look and feel" when switching between the Domino and WebSphere environments. You also need to merge "security domains" as described later in this section.

There are a number of design considerations to bear in mind when implementing browser integration. For example, by making calls (HTML links) between servers, each server should have fully qualified links to the other (global properties). Each server needs to pass all relevant information to all others because WebSphere would not know much about the user when coming through Domino. It’s vital to tell the other party how to return, so give it a full URI (document ID and so on). And at a minimum, it’s very important to define proper error handling in a way each party understands. Another point is that fields on a Domino form need to be editable for sending the information the user entered.

WebSphere tracks user and session information for preventing duplicate entries (HTTPSession object). Domino does not track user state, which occasionally results in conflict documents. Therefore, you need a way to generate unique documents and/or references for WebSphere to prevent duplicate posts. To do this, you need to install the WebSphere plug-in that supports the remote communication (HTTP v4.0) between each server in Domino. And you must merge the security domains of Domino and WebSphere. This means WebSphere single sign-on (SSO) must be installed in Domino, all servers must be in the same DNS domain, and Domino and WebSphere must use the same LDAP directory. **(Note: WebSphere Application Server Advanced Single Server Edition v4.0 does not support SSO.)**

**Domino to WebSphere integration**

Domino to WebSphere integration consists of two flavors: Domino to the Web container and Domino to the EJB container. Each is described in this section.

**Domino to the Web container**

You can connect Domino to the Web container three different ways. In the first, Domino acts like a Web browser or client. This can be done via sockets, which is very flexible, but requires a large amount of work. This can also be done through a URL or URL Connection class.
The second and third methods Domino can use to connect to the Web container include the HTTP programming model. HTTP is a request-response application protocol in which the parameters of the request must be set before the request is sent. There are two ways to send data to a servlet by the HTTP server: GET and POST. These methods determine how the data is sent to the server.

For the GET method, the input values are sent as part of the URL in the QUERY_STRING environment variable. When using HTTP GET, there are limitations in security (the input variables will be sent as part of the URL) and the amount of data that can be sent (many input values will result in a very long URL). There is no direct definition of the size limit; it's in the range of 255 bytes up to 8 KB. If this limit is exceeded, the server should send an error 414 (Request-URI Too Large). Additionally, the data has to be encoded on the request URL. This encoding can be either simple (translate single characters into other characters, for instance, a space into â+) or complex (translate special characters into three characters % hex hex and so on). For example, using UTF-8 as the encoding scheme the string "The umlaut ü@foo-bar" is converted to "The +umlaut+%C3%BC%40foo-bar" because in UTF-8 the character Ä¼ is encoded as two bytes C3 (hex) and BC (hex), and the character @ is encoded as one byte 40 (hex). The HTTP GET can be used in LotusScript and by Java agents.

For the POST method, data is sent as an input stream and its length is saved in the CONTENT_LENGTH variable, which supports an unlimited amount of data because the input values are not sent as part of the URL, they are sent as an input stream in the HTTP body. Therefore, you can send more data with the POST method. In addition, the data doesn't have to be in plain text as with the GET method. Unlike GET, the "content type" defines the type of data and encoding and can only be used by Java agents. When sending to WebSphere, you should format fields as individual fields (similar to GET) and send serialized Java objects (Beans).

**Web design considerations**
When connecting Domino to the Web container, you need to acquire authentication credentials. (Client authentication data can be extracted from the server or header.) LotusScript cannot set cookies in the GET request, but it can set user name/password information and send it to the Web server as basic authentication. The user credentials (SSO token) could expire before communication happens (if saved/queued).

So what do you do if there’s a server outage? LotusScript can call WebSphere immediately and reply, but Java agents cannot access documents in the UI. That means that you must be able to read documents in the database, to queue requests, to trigger the problems, and to inquire about the status of the Java agent in a user friendly way. At least batching multiple requests is more efficient and can process more data!

Bear in mind that every document is its own HTTP connection/request/response, regardless of whether or not an agent processes multiple records (documents). Before you can access a rich text Notes item, you have to save it.

**Domino design considerations**
Of course, Domino must run the Web task for LotusScript. This task catches requests and adds a unique suffix to get a new response. Basic authentication (a login dialog box in the browser
window that forces the user to enter a valid user name and password) is supported by LotusScript and Java agents (and only Java agents support SSO tokens).

**Domino to EJB container**

To connect Domino to an EJB container on WebSphere, you must use RMI/IIOP for communication. An InitialContext must be used for accessing the EJB home reference via JNDI (Java Naming and Directory Interface).

**Domino R5**

IBM's WebSphere Application Server Enterprise Edition is the only version that allows standard CORBA; Domino R5 does not meet this criteria. Therefore, you must use an intermediary server. This intermediate server is a "normal" EJB client program; it meets all WebSphere requirements and acts as a proxy for Domino. Domino R5 communicates with the intermediate server via HTTP (to the Web container), RMI, and sockets:

![Figure 3. Domino 5/WebSphere server architecture](image)

But there are some WebSphere limitations in this configuration. The J2EE spec stipulates that servlets cannot be RMI servers; they can only be RMI clients. Also EJBs cannot start their own threads and cannot be RMI servers. However, WebSphere allows for a "generic" server. This can be any type of program started and stopped via the console (although not in WebSphere Application Server Single Server Edition). We recommend that you run intermediate servers on each WebSphere server.

Communication from Domino R5 to the intermediate server can use any protocol; RMI is preferred by Java agents only. Communication also needs to have references to multiservers; intermediate servers will have references to all EJB containers. If you must talk to servers behind a firewall, use RMI/IIOP instead of RMI/JRMP.

Intermediate servers should run on the same node as WebSphere. Each EJB application should run one copy, which can be defined as a generic server in your WebSphere Application Server domain. This allows for remote starting and stopping.

**Domino 6**

When connecting Domino 6.x with WebSphere Application Server 5, you don’t need an intermediate server:
WebSphere to Domino

The connection from WebSphere to Domino can be done in two ways: HTTP (similar to Java agents to WebSphere) and Java/CORBA.

WebSphere to Domino HTTP

To connect WebSphere to Domino HTTP, use URL and URL Connection classes. This means WebSphere mimics a browser or uses the Domino servlet engine (and a Domino servlet can transfer anything in the response). Domino, on the other hand, only accepts forms by post. Here you have to avoid using buttons or action links because it’s difficult to mimic their posts. These forms should forward/process the request to other forms/pages/views via a view where you have to select the form property "Treat document content as HTML." Then you have to create a view that shows selected documents and add a $$Return field to the form with the formula "[ $$ + @SubSet(@DbName; -1) + ViewName + $$ + docID + $$?OpenDocument]."

WebSphere to Domino classes

You can use standard Domino classes and techniques (import to your code "lotus.domino."). Domino itself sees them as stand-alone Java programs. The session object can be created from NotesFactory ("static public Session create Session (String host, String user, String passwd")), but do not use NotesThread.

In a Notes session, you need to establish a separate session for each request. You could use a Web container, but it needs its own way of synchronizing and limiting threads. EJBs are single-threaded, and each thread in the pool can have its own connection. (In general, J2EE 1.3 disallows thread "ownership." If you need this feature, use asynchronous beans available in WebSphere Application Server 5.x.) Domino 5 and earlier cannot handle a very high rate of CORBA connections; this causes the connections to fail. However, the CORBA implementation of Domino 6 is much improved. Java must recycle objects if you want to avoid Domino server crashes caused by running out of memory.
Summary

As you have seen, there are many ways you can connect Lotus Domino and IBM WebSphere, depending on the solution you want to build. Therefore, it's absolutely necessary to design your enterprise application to meet all requirements. If Domino is your central architecture, then your Domino server is the HTTP server, application server, and data store. If Domino is part of your J2EE environment, then your Domino server is part of the data store and is used through Domino objects.

© Copyright IBM Corporation 2004
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)