CHAPTER 7

What to Put on WAS vs. Domino

"'Domino and WebSphere: So Happy Together.' Both Domino and WebSphere are thriving because they solve different customer problems."

—*E-Pro Magazine*, June 2001

When looking to combine features of both Domino and WAS (or J2EE) in a single application, designers and developers often find that some application elements could be supported on either WAS or Domino. This chapter explores those areas where WAS and Domino offer comparable function and tries to provide some insight as to which platform may be more appropriate for a particular set of requirements. Although there are certain cases where there is a clear choice of where best to place a function, it is our experience that there are also cases where there is no clear choice in terms of function alone, and other factors such as development skills or familiarity may be the basis for the final choice. In fact, these other factors, especially development skills, may override any purely functional benefits.

At a high level, the type of application is an important factor in deciding on which platform to build. As discussed in earlier chapters, the strengths of Domino lie in collaborative applications. Domino is good at applications involving user interaction, providing a scalable document data store, rich content handling, and a robust security model. WebSphere Application Server (WAS) is ideal for transactional applications, highly leveraging a relational data store, and providing enterprise-level J2EE application hosting. WebSphere Portal excels at application and data aggregation, providing a rich framework for Web-based applications.

Assuming that we have an application that has both collaborative and transactional elements, let's review some of the key features of each platform to help understand which parts may be supported by which platform.

Lotus Domino Server

Lotus Domino is a comprehensive application platform for collaboration that handles both connected and disconnected requirements for data and applications. Many customers initially purchase Lotus Domino for the built-in enterprise e-mail, calendar, and scheduling applications, making those types the most-widely deployed collaborative applications. The majority of customers, however, exploit the document-sharing and form capabilities that support core business processes, which enable employees to work together efficiently and securely. Domino is comprehensive; it provides the complete infrastructure needed to create, test, deploy and manage distributed, multilingual applications including directory, database, application server, administration, security, connectivity, Web server, e-mail server, calendaring engine, and so on—all in one system.

WebSphere Application Server

The WAS is IBM's core product for J2EE application support. IBM has differentiated WAS from other J2EE application servers in its scalability. From its inception, WAS was geared toward large-scale J2EE applications. It provides features supporting highly distributed applications and their management and support for the latest J2EE and Web services standards. It also addresses scalability and performance with support for deployment across multiple servers, load-balancing services, and automated performance tuning. Today, WAS is available in several configurations across a wide range of platforms, allowing any size of J2EE application to be deployed in the most cost-effective way possible.

In contrast to Domino, WAS does not contain its own data manager. Instead, it relies on a relational database being available in the environment for application data storage or for J2EE components themselves, such as entity EJBs. One way that WAS provides access to those data sources is by leveraging the J2EE Connector Architecture (JCA). JCA defines a set of service contracts that a connector developer can expect will be available to the adapter at application runtime. The three basic services defined in JCA V1.0 include connection management, transaction management, and security management.

Also unlike Domino is the fact that WAS relies on external sources for user directory (or what it terms "user registry") functions. For WAS (and J2EE), the user registry is required for user authentication relative to providing secure access to application components. WAS can be configured to use two categories of user registries—local; for example, the Windows Active Directory or Unix user account file (/etc/passwd), or LDAP; that is, directories supporting the Lightweight Directory Access Protocol. The fact that Domino can provide LDAP access to its directory enables it to be considered for the WAS user registry. For single sign-on (SSO) support between Domino and WAS, and LDAP user registry must be used. We discuss the options for directory configuration in detail in Chapter 12, "Security and Single Sign-On."

Domino Features Enabling Applications to Integrate with J2EE

Domino ships with features that facilitate the integration of Domino data with J2EE applications. You can reap many benefits by combining Domino and J2EE applications including, most importantly, the addition of human interaction into scalable and transactional applications built on J2EE. When integrating Domino with a J2EE-based application, there are two key ways to access Domino functions:

- Domino Objects for Java
- Domino Custom JSP tags

The Domino Objects for Java are essentially the Domino Java API. These objects allow for Java object-based access into the Domino data store for the key database functions of create, read update, and delete—or CRUD actions. The Domino Objects for Java API allows access not only to Domino data but also to key Domino services, such as registering users, running agents, sending mail, and others. The Domino Objects for Java can be used by experienced Domino developers for adding Domino specific function to J2EE applications. For example, a servlet or JSP can be enhanced with Domino function by using the Domino Java Objects within the servlet's or JSP's Java code.

The Domino Custom JSP tags are for use from within a JSP. These tags are XML tags embedded in a JSP and provide access to Domino data, data validation, and flow control. The Domino Custom JSP tags are built on the Domino Objects for Java and are intended for use by experienced JSP and JavaScript developers. These developers can use the Domino tags to build a J2EE application that uses Domino data and services, without having to become familiar with the pure Java Domino APIs.

Lotus Domino Toolkit for WebSphere Studio

Starting with Domino 6.0.2, the Lotus Domino Toolkit for WebSphere Studio provides developers with an intuitive tool to implement the Domino custom JSP tags into their WebSphere applications. The toolkit is a set of plug-ins that enhances the WebSphere Studio Application Developer (WSAD) user interface. It enables J2EE developers to integrate Domino functions into Web applications without the complexity of the Domino Java API or even a detailed understanding of the Domino Custom JSP tags.

The toolkit adds a new view to WSAD, a "Domino view," which allows you to work easily with Domino database objects such as forms, fields, views, and agents. Using this new Domino view within WSAD, you can access a Domino database (locally or remotely) and display the database's forms, fields, views, and agents. These objects can be dragged and dropped onto JSPs being developed, and the equivalent Domino custom JSP tag code is inserted. Using the Lotus Domino Toolkit enables developers without an in-depth understanding of Domino function to easily incorporate that function into their applications.

XML

Domino has extensive XML capabilities that can be used from a J2EE application. A J2EE application can use the Domino Java API to retrieve Domino documents in XML format without knowing specifics about the data stored in Domino. Then, the J2EE application can process the XML and transform it to the specific format needed by the application. If the application needs to update the Domino data, the same interface can be used to update the database by passing back XML.

Web Services

Emerging Web services standards simplify and broaden application integration by defining a standardized means to make service requests and receive response data over common Internet protocols. The lingua franca of Web services is the XML based access protocol known as Simple Object Access Protocol (SOAP). Designers express the application or service interface using XML. Domino can host Web services that expose Domino data and function using Java or LotusScript. Using a combination of a J2EE server such as WAS, appropriate SOAP classes and the Domino Java Objects, developers can expose desired portions of their current Domino applications as Web services. To do so requires some Java development skills and knowledge of the Domino object model, which we discuss in detail in following chapters. For pure J2EE applications, WAS ships with the needed SOAP classes, and the WebSphere Studio development environment has wizards for creating, consuming, managing, and deploying Web services.

Domino Data in a Relational Database

Domino has a capability, via the Lotus Enterprise Integrator (LEI) feature, to store all document data in a relational database, such as DB2, Sybase, or Oracle, rather than in a Domino NSF database file. The benefit is that Domino maintains its security controls and the database design, but since the document data is stored in the relational database, document access is improved due to the performance efficiencies offered by the relational database. This access method is referred to as Advanced RealTime connectivity. Advanced RealTime connectivity allows significantly better scalability and performance, since the relational technology is used for data storage versus the Domino NSF file format.

The Advanced RealTime connectivity feature can be used as a means to integrate Domino data with J2EE applications. As mentioned previously, J2EE applications running on WAS can access relational data stores directly using JDBC or EJB interfaces. Instead of using Domino provided mechanisms to access Domino data stored in NSF files, the application can utilitize relational data access methods. A designer can achieve high performance by employing standard data access techniques such as connection pooling, caching, and minimizing JNDI lookups.

J2EE allows for the abstraction of databases and their connections from application code by accessing a business object layer rather than the database directly. This layer, in turn, accesses the database itself. The same business objects can be accessed by different applications. The database design is decoupled from the application itself, creating easier maintenance going forward.

WAS (J2EE) Functions Compared to Domino

In this section, we discuss the main areas where WAS and Domino overlap in function and present the conventional wisdom (and our view) about using one over the other.

Servlets vs. Web Agents

At a high level, J2EE servlets and Domino Web agents are pretty much equivalent—they provide a means to run server-side programs triggered by client-side (browser) events. The Domino "Web agent" function is new with Domino 6 and basically extends the traditional Domino agent function to include triggering from Web events. The traditional Domino agent was restricted to server-side programs triggered by time or by Domino database events. Both servlets and Web agents can return HTML to the client.

Domino Web agents can be coded in either Java or the Domino-specific languages of LotusScript, formulas, or simple actions. In fact, Web agents are the only context in which LotusScript may be used for Web applications. Thus, Web agents may be easier to implement for Domino developers. The Web agent, since it runs as part of the Domino server, has direct access to Domino data elements. The biggest drawback to using Web agents for server-side functions is that the agent must be loaded each time it is invoked. This is done by the Domino agent manager, which does not provide for agent caching. For Web agents written in Java, an entire JVM must be started to run the agent. If the design calls for frequent use of the agent, loading on each invocation usually leads to unacceptable performance.

Servlets, on the other hand, are architected to be run efficiently by the J2EE application server. Servlets are loaded either on server start or on first invocation of the servlet and are simply passed subsequent requests by the application server. If the servlet function needs to access Domino resources, this can be accomplished via the Domino Objects for Java. The servlet designer has a choice of how Domino is accessed, which can be either locally (where Domino is running on the same server as WAS) or remotely. Naturally, there is overhead in accessing Domino remotely, but this can be tailored so that the access is not needed for each servlet invocation. We discuss strategies for accessing Domino via Java and J2EE in more detail in Chapter 10, "Accessing Domino from J2EE."

Which to choose then? We usually adhere to the common guideline of implementing server-side functions as servlets, especially when the functions will be invoked frequently. If the functions are infrequent and require a fair amount of or sophisticated Domino processing, then Web agents are appropriate.

Java Server Pages (JSPs) vs. Domino Forms

Another area of overlapping function between WAS and Domino is the ability to generate and process HTML content, especially forms. A designer can include forms in a Web application by using either JSPs managed by WAS or Domino database forms served by the Domino HTTP engine. A third possibility is available now with the support for Domino Custom JSP tags, where any Domino database form can be converted into a JSP and handled by WAS.

The choice between WAS and Domino here is not clear-cut. There are several factors that could affect the decision. Probably the most influential ones are the following:

- **Overall application look and feel.** If the application is predominantly JSP- or Domino-based, then it makes sense to stay with the same rendering. Although if there is any JSP content, it may be easier to convert Domino Web content to JSPs rather than the other way around.
- **Development skills.** If the development experience lies with one product, then it will likely cost less to develop using that product.
- **Unique feature rendering.** There are certain cases where attempting to convert a Domino form to a JSP will not be of perfect fidelity. This can occur when using the Domino Custom JSP tags. (In Chapter 10, we discuss ways to generate JSPs from Domino forms with greater fidelity.)

WAS vs. Domino Servlet Engine

Before the advent of WAS, the Domino server product provided a means for plugging in a servlet engine to handle servlets in addition to the standard Domino resources. So, there is the possibility of using a servlet engine under Domino versus WAS for servlet processing. The choice here is clear-cut. WAS should handle all J2EE resources. It's difficult to concieve of any advantage to having Domino handle servlets.

Keeping an Eye on the Future of Domino and WAS

For IBM, the future of both WAS and Domino is clearly all about how best to integrate the product families in terms of both function and developer communities. Clearly with recent product versions, this process has started. How may this integration affect application design decisions that need to be made today? We consider this question in the following section.

Toward Open Standards

A key issue in the Web application server and services marketplace is the choice between the use of open standards vs. proprietary programming interfaces. Or, to be blunt, J2EE versus .NET. As is well documented, an open standards foundation is a major part of IBM's strategy in the marketplace. WAS, being an implementation of J2EE and now Web services standards, will continue down the open standards road. Domino, having a longer product history, will continue to incorporate open standard interfaces.

Although proprietary technology is an important differentiator in the market, even Lotus has long supported key Internet standards. Domino delivers enterprise messaging while supporting an open standards-based application development architecture that includes Java, JavaScript, XML, HTML, LDAP, SMTP, POP3, IMAP, and SSL. Domino has embraced Java as both an

external interface, providing Java classes for all its key interfaces, and as an internal language that can be used in place of Lotus' BASIC-like programming language, LotusScript. It is natural to expect that the integration of the WAS and Domino products will occur along these open standards lines.

Browser-Based Clients

Another sign of the impact that the Web application model is having on Domino are the iNotes and, more recently, the Lotus Workplace products. Traditionally, Lotus Notes was a client/server architecture based on the feature rich Notes client and the Domino server. With Domino R5 and iNotes, browsers became a prominent client. Lotus iNotes for Web Access, despite its browser dependencies, was an important step along the path to provide Notes client function via the Web browser. The Lotus Workplace product is the latest step along this path.

Browser-based client function is important for applications that integrate several, often disparate types of function on a single browser screen—the so-called Web portal. For example, using the WebSphere Portal Server product, you can integrate transaction processing under WebSphere with e-mail, calendar, and other Domino functions. Thus, you can now construct a Web application that looks and feels like an extension of the iNotes application, resulting in a single, browser-based desktop from which users can access all the applications that they use on a daily basis.

It is likely that IBM will continue to provide client function via the browser and enhance the WAS and Domino functions that can be presented in a portal context.

Convergence of Function

The integration between WAS and Domino also will occur with functions that span both products. Security is the prime example. IBM has implemented single sign-on (SSO) support between Domino and WebSphere. SSO lets users authenticate once when accessing both Domino data (databases, documents, fields) and WAS resources, such as HTML, servlets, JSPs, and EJBs. This is accomplished via sharing an encryption mechanism used for passing user credentials and sharing a common user registry using LDAP.

Another area of common function is data storage. We can see that Domino's data storage technology (the NSF database) has already benefitted from synergy with other IBM database products, especially DB2. Domino R5 provided faster recovery and improved reliability for its data store using transaction logging. We can expect this sort of synergy to continue, perhaps resulting in convergence of Domino's document data store with the relational model.

IBM and Lotus will continue to meld their product families into one. Lotus K-station (Domino's portal server) has already been merged into WebSphere's portal server as a single portal solution with the benefits of both products. IBM's recent move to allow WebSphere workloads on its iSeries Dedicated Servers for Domino (DSD) line of servers is a good indication that IBM understands how important this integration is.

Another notable item is the Domino Collaboration Objects for Java. IBM and Lotus say their intent is not to extend Domino but to provide a more intuitive interface to Domino objects with fewer Java methods and classes. The current array of objects includes the following:

- CalendarEntry (to search calendars and create new entries)
- Mail (to compose and send e-mail)
- Login and workflow initiation services

As for Domino's traditional programming environment, don't assume IBM will abandon LotusScript. Lotus Notes and Domino have an enormous customer base of users and developers using LotusScript-based applications. Continued support for this language is unquestionable. As new classes and properties are developed for LotusScript, equivalents for Java will follow. Obviously, the intent is that J2EE (Java) programmers can be trained to work with Domino without having to learn a new language.

Evolution of WebSphere

So, if the future of Domino is WebSphere, what's the future of WebSphere? Since WAS is IBM's implementation of the J2EE standard, the evolution of J2EE will largely determine what WebSphere becomes.

IBM is committed to building the most complete and best-performing application server based on the J2EE standard. And because many areas aren't covered by the standard, IBM can be creative and add value by supplying add-on features that make WebSphere more enterprise ready. IBM is also adding application products and proprietary extensions to the base WebSphere product to meet needs expressed in the marketplace.

IBM updates WAS every six months or so with a major release. Each version of WebSphere gets a little easier to install and manage, but the real activity is in the extensions and applications. Consider the following:

- With WebSphere MQ, IBM implements the Java Messaging Services (JMS) standards. Numerous connectors to CICS and other legacy systems and middleware are also planned or shipping.
- IBM has rewritten Net.Commerce into Java to run under WebSphere as the WebSphere Commerce Suite—a high-powered storefront system.
- IBM has grafted speech recognition technology (as well as its vaunted national language support) onto WebSphere.
- IBM has implemented portals, wireless support, and B2B capabilities as Java applications that run in the WebSphere environment.
- IBM is moving its legacy connectivity products to the WebSphere environment with offerings such as WebFacing on the iSeries and Host Publisher.
- IBM has leveraged Lotus' expertise with its ERP Connectors and introduced several WebSphere adapters that are based on Lotus LEI connector technologies.

WebSphere is the logical place for IBM to create new applications because they can be leveraged over the entire eServer line. Expect a steady stream of WebSphere-based products from IBM over the next few years. Watch areas like content management, system management (Tivoli is already starting to play in these areas with its Policy Manager), collaboration, and application accelerators.

Tooling for developing WebSphere applications is also advancing rapidly with the new WebSphere Studio products. Based on the open-source Eclipse project, IBM has replaced VisualAge for Java and the old Studio product and added new support for XML and Web services. For some of the IBM hardware platforms, such as the iSeries, IBM is adding platform-specific extensions to aid in application development and integration.

A big question is the Lotus Domino Designer client. It's hard to see how IBM will merge this with the WebSphere tooling. You may see, however, improved Domino deployment options for objects developed in the WebSphere Studio tools.

IBM has already repositioned several product lines and placed them under the WebSphere umbrella. Is Domino next? Will we see a WebSphere Mail and Collaboration Server? Assuming that there will be a new integrated WebSphere/Domino Server in the future, it will be worthwhile to keep an eye on WebSphere and J2EE evolution.

A Web Conferencing Example Using Both Domino and WAS

In this section, we discuss the design of a project that provides an "on demand" Web conferencing service and that incorporates both Domino and WAS product functions. It illustrates various ways in which Domino and WAS-based products can be combined to build applications providing complex function without a lot of custom application development.

The basic function of the service is to allow anyone with Internet access to register, schedule, and join browser-based conferences. The conferences include chat, multi-media, and whiteboard sharing functions. The Domino-based Lotus Instant Messaging (née Sametime) Room Server product is used for hosting the Web conferences. To register the Web conferences, the WAS based WebSphere Everyplace Subscription Manager (WESM) product is used. The Lotus Enterprise Meeting Server (EMS) product, which is a J2EE application, thus WAS-based, is used to manage the meetings and perform load balancing for the Sametime room servers. The EMS keeps track of logging into and exiting meetings for registered users. The Sametime room servers run on Domino servers, as they don't require transaction capability. The user registry is provided by the IBM Directory Server product and is shared across the room, subscription, and meeting servers. There was also a requirement for an outbound mail function, and this was provided by a stand-alone SMTP server. The project was designed for 24×7 availabilty, so many of the server nodes were built as clusters using an IBM clustering product, HACMP.

Figure 7-1 shows the overall architecture of the service and where the different types of servers are placed in the network tier layout. We point out that due to product requirements, not all the WAS servers are at the same WAS version. WESM runs on WAS V5, but the EMS product supports only up to WAS V4. For this reason, the WESM and EMS servers were built on separate server systems.

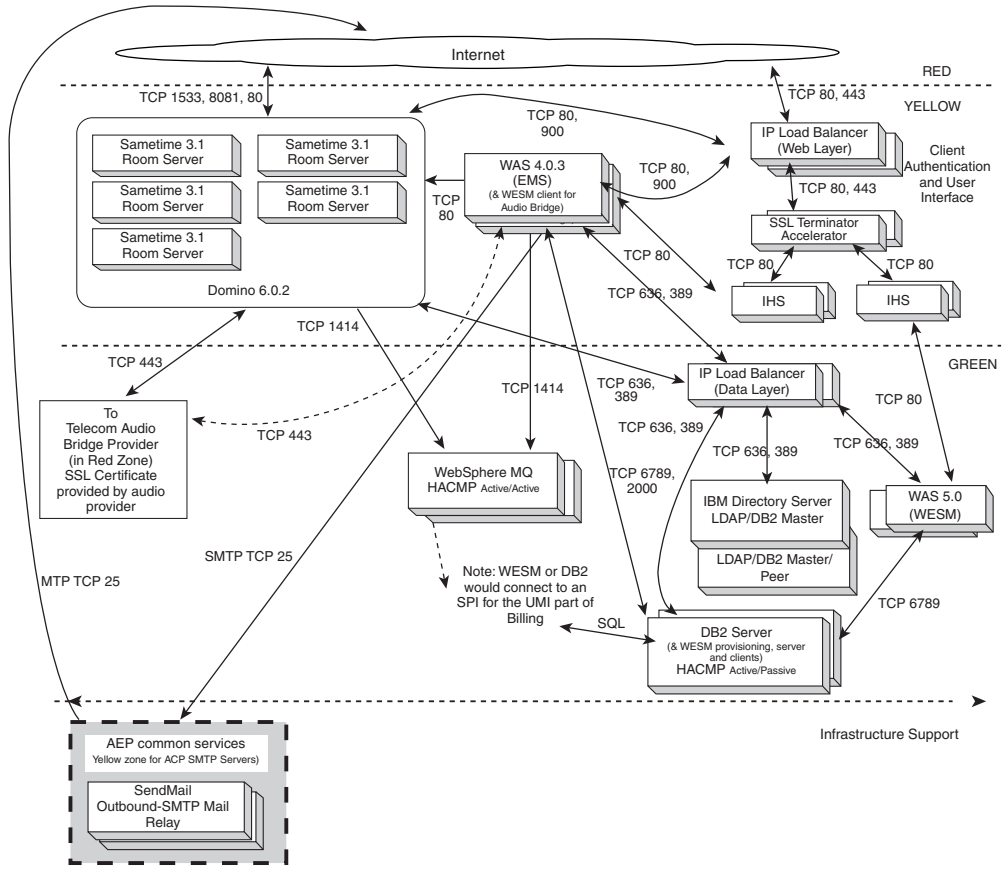

Figure 7-1 Web Conferencing logical server architecture.

Directory Master-Peer Design for High Availability

The directory function of the Web conferencing service had two key requirements—that it be shared among the functional servers (e.g., for single sign-on) and that it be highly available. The approach taken was to place the directory server behind a shared load balancer and to make use of a master-peer replication topology provided by the IBM Directory Server product. Figure 7-2 depicts the directory server configuration.

In the master-peer approach, the directory client will read/write to the master. If the master server fails, the client requests are directed to the peer server until the master is made active again.

The replication process keeps the data in the master and peer directories synchronized. Whenever a write (or update) is to be done against the directory, it goes to the master. After the write/update is complete, it replicates it to peer server immediately. When the master server is down, the write/update goes to peer server, and it will be queued for replication once the master is restarted.

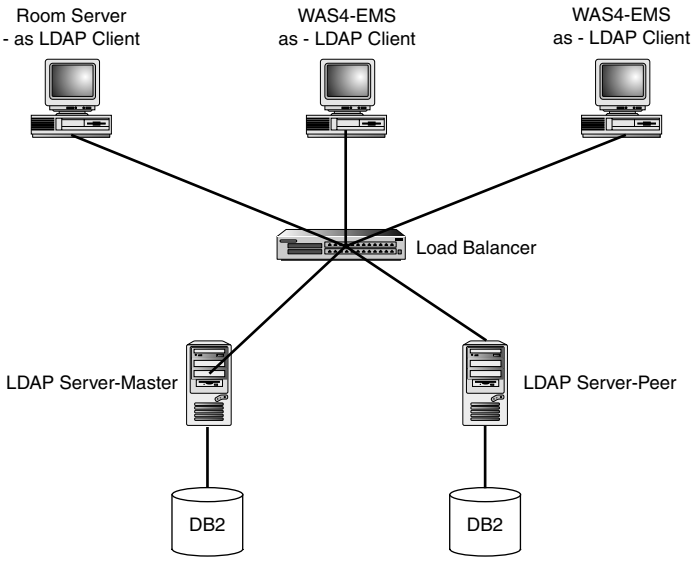

Figure 7-2 Design for the shared directory.

For directory sharing, the Domino, WESM, and EMS servers are configured to the virtual IP address of the directory server. As part of their user management functions, the EMS and WESM applications perform write/update operations to the directory server.

WebSphere MQ Application Details

EMS communicates with the Sametime servers using the Java Message Service (JMS). This JMS communication is implemented using IBM WebSphere MQ message queuing. The JMS communication supports load balancing, server independence, and failover necessary to provide central meeting management for all of the clustered Sametime servers. It also allows centralized logging, statistics, and configuration change notification.

The Sametime Room servers must be able to communicate to EMS/WAS using TCP/IP port 900 (WebSphere 4.x JNDI port). The EMS and Sametime Room servers must be able to communicate to WebSphere MQ using TCP/IP port 1414.

WebSphere MQ Cluster Design

Because the WebSphere MQ (WMQ) server acts as the blood supply of the service, availability is a key criterion for these servers. It is also important that the design allow for the easy addition of MQ servers. IBM's High Availability Cluster Multi-Processing (HACMP) product (for AIX) is a control application that can link servers into highly available clusters. Clustering servers enables parallel access to data, which can help provide the redundancy and fault resilience required for business-critical applications.

The WMQ clusters reduce administration and provide load balancing of messages across instances of cluster queues. They also offer higher availability than a single queue manager because following a failure of a queue manager, messaging applications can still access surviving instances of a cluster queue. However, WMQ clusters alone will not provide automatic detection of queue manager failure and automatic triggering of queue manager restart or failover. HACMP clusters provide these features. The two types of cluster can used together to good effect.

By using WMQ and HACMP together, it is possible to further enhance the availability of the WMQ queue managers. With a suitably configured HACMP cluster, it is possible for failures of power supplies, nodes, disks, disk controllers, networks, network adapters, or queue manager processes to be detected and automatically trigger recovery procedures to bring an disabled queue manager back on-line as quickly as possible.

An HACMP cluster is a collection of nodes and resources (such as disks and networks), which cooperate to provide high availability of services running within the cluster. Hopefully, we've been making a clear distinction between such an HACMP cluster and a WMQ cluster, which refers to a collection of queue managers that can allow access to their queues by other queue managers in the cluster.

A "mutual takeover" configuration is one in which all nodes are performing highly available (movable) work. This type of cluster configuration is also referred to as "Active/Active" to indicate that all nodes are actively processing critical workload. An Active/Active cluster configuration is used for the Web conferencing service.

The load balancing is based on client connections and not the message distribution. Each Sametime room server has a connection to either of the cluster nodes through connection channel. If the queue exists on the queue manager node to which it is connected, it will always go there for the duration of the connection. Figure 7-3 depicts the overall cluster design.

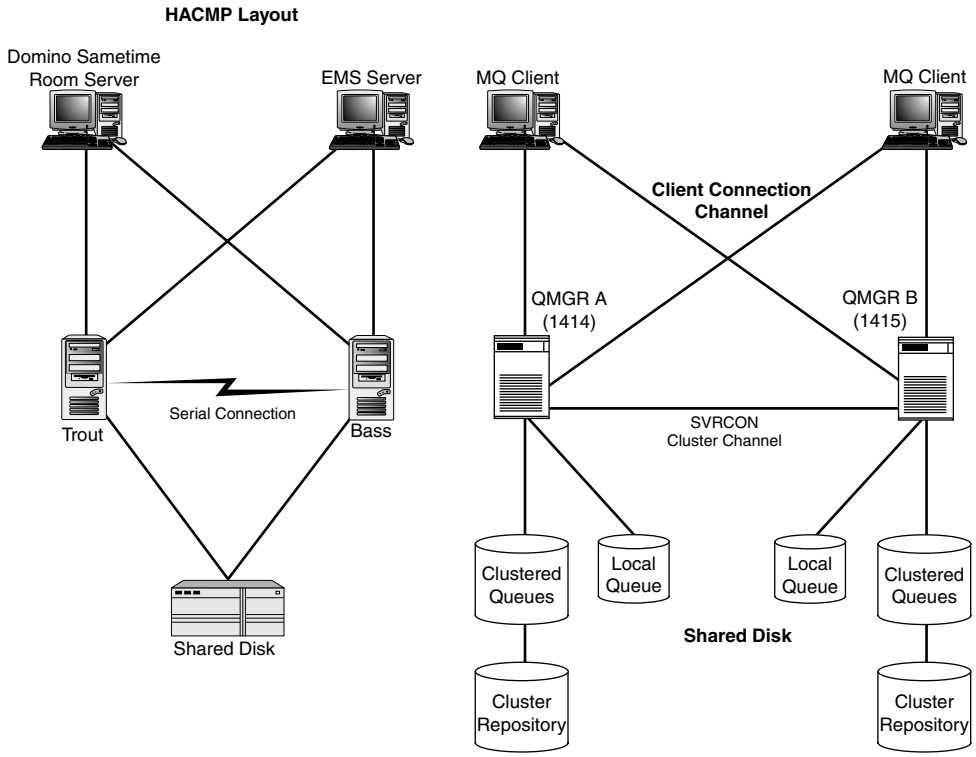

Figure 7-3 WebSphere MQ HACMP cluster (active-active) design.