DISTRIBUTED APPLICATIONS VS. INTEGRATION AND COMMERCIAL MESSAGING SYSTEMS

This article is about enterprise integration—how to integrate independent applications so that they can work together. An enterprise application often incorporates an n-tier architecture (a more sophisticated version of a client/server architecture) enabling it to be distributed across several computers. Even though this results in processes on different machines communicating with each other, this is application distribution, not application integration.

Why is an n-tier architecture considered application distribution and not application integration? First, the communicating parts are tightly coupled—they dependent directly on each other, so that one tier cannot function without the others. Second, communication between tiers tends to be synchronous. Third, an application (n-tier or atomic) tends to have human users that will only accept rapid system response.

In contrast, integrated applications are independent applications that can each run by itself, but coordinate with each other in a loosely coupled way. This enables each application to focus on one comprehensive set of functionality and yet

delegate to other applications for related functionality. Integrated applications communicating asynchronously don't have to wait for a response; they can proceed without a response or perform other tasks concurrently until the response is available. Integrated applications tend to have a broad time constraint, such that they can work on other tasks until a result becomes available, and therefore are more patient than most human users waiting real-time for a result.

Commercial Messaging Systems

The apparent benefits of integrating systems using an asynchronous messaging solution have opened up a significant market for software vendors creating messaging middleware and associated tools. We can roughly group the messaging vendors' products into the following four categories:

1. *Operating Systems.* Messaging has become such a common need that vendors have started to integrate the necessary software infrastructure into the operating system or database platform. For example, the Microsoft Windows 2000 and Windows XP operating systems include the Microsoft Message Queuing (MSMQ) service software. This service is accessible through a number of API's, including COM components and the System.Messaging namespace, part of the Microsoft .NET platform. Similarly, Oracle offers Oracle AQ as part of its database platform.

2. *Application Servers.* Sun Microsystems first incorporated the Java Messaging Service (JMS) into version 1.2 of the J2EE specification. Since then, virtually all J2EE application servers (such as IBM WebSphere, BEA WebLogic, etc.) provide an implementation for this specification. Also, Sun delivers a JMS reference implementation with the J2EE JDK.
3. *EAI Suites.* Products from these vendors offer proprietary—but functionally rich—suites that encompass messaging, business process automation, workflow, portals, and other functions. Key players in this marketplace are IBM WebSphere MQ, Microsoft BizTalk, TIBCO, WebMethods, SeeBeyond, Vitria, CrossWorlds, and others. Many of these products include JMS as one of the many client API's they support, while other vendors—such as SonicSoftware and Fiorano—focus primarily on implementing JMS-compliant messaging infrastructures.
4. *Web Services Toolkits.* Web services have garnered a lot of interest in the enterprise integration communities. Standards bodies and consortia are actively working on standardizing reliable message delivery over web services (i.e., WS-Reliability, WS-ReliableMessaging, and ebMS). A growing number of vendors offer tools that implement routing, transformation, and management of web services-based solutions.

Source: http://www.enterpriseintegrationpatterns.com/patterns/messaging/Introduction.html