Chapter V

Enterprise Electronization and Integration

STRATEGY AND RATIONALE

The goal of the electronic enterprise is to implement all major applications to build the extended enterprise that functions as a paperless organization, whose units and workers process information and communicate via all layers of the Enterprise Information Infrastructure (look at Chapter 3).

The strategy of the e-enterprise development is the integration of all business and application components via the Internet or intranet/extranet environment ("electronization"). The e-enterprise evolves from the net-commerce, based on EDI (1980's/1990's), the e-commerce stage (1997) and its follower - the e-business stage (1999), as a result of the electronization and integration of all enterprise applications.

The development of the e-enterprise began in the 2000's with the re-engineering of ERP systems in two directions:

- The expansion of ERP into links with SCM and CRM systems,
- The application of the four-tier IT architecture, where the Web-driven server integrates all components of the above systems.

Of course, the development of a complete e-enterprise will take decades, if not the whole 21st century, having in mind the whole economy. This process involves about 15% of the American GDP, which means that spending on

information management exceeds $1 trillion. This sum can transform the industrial model of the enterprise into the information model quite effectively.

The annual productivity growth in the American Economy in the 1990's was at the level of 3% to 5% and was mostly influenced by information technology. As a result, almost each American business is almost transforming itself into the advanced computerization stage, which is reflected in 400,000 IT jobs vacancies in 2000/2001. Another result of this trend is the restructuring process of many corporations, which in the 2000's is reducing employment by thousands of workers, who in most cases are being replaced by the job automation and informatization.

The e-enterprise is not longer *science fiction* but the reality of the American business landscape.

SYSTEMS ELECTRONIZATION

The rapid electronization of an enterprise's applications in the 2000's is triggered by the application of the Internet, extranets, and intranets' capabilities, and particularly by their Web technology. The latter provides the following solutions:

- World Wide Web (WWW), whose heart is hypertext, by which information is organized as a series of documents with links for search and retrieval of text, images, sound, and video,
- Graphic User Interface (GUI), which is based on home pages whose users are impressed by its friendliness and ease of use,
- Web browsers that are powerful tools in searching information and fulfilling computing tasks,
- Web software servers that are computer language independent to allow for the integration of different applications that up until now have been isolated and have created the so-called "island of automation,"
- Enterprise Information Portals that integrate the user's community with a variety of shared information and applications,

- Easy and rapid connectivity among users and applications through the Internet and its derivatives, such as extranets and intranets,

- Internet Service Providers (ISP) that provide access to the Internet in almost every city and town, allowing a user paying a monthly fee to access the Web and enjoy a sophisticated e-mail system and plenty of other services.

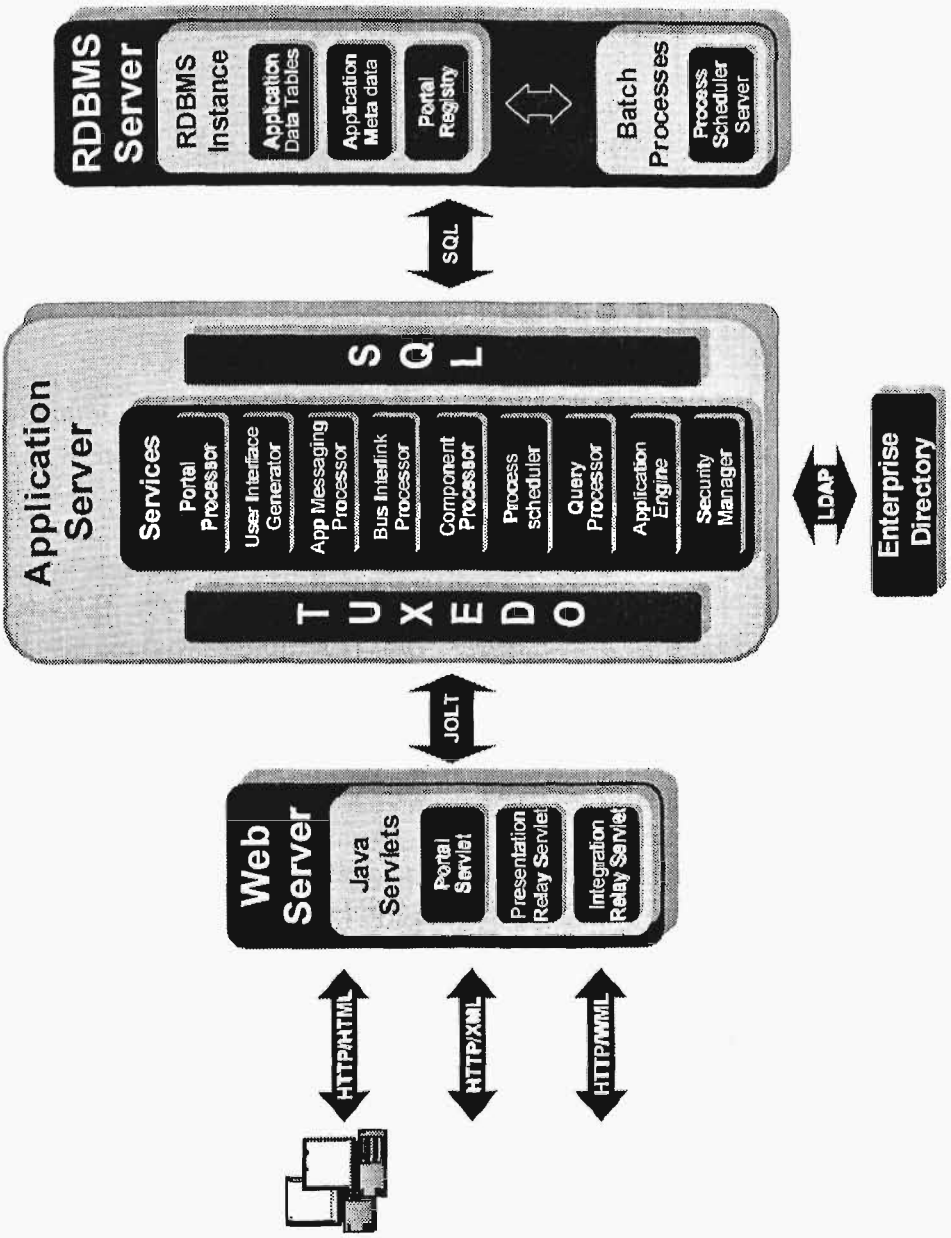
Discussions about internet (Web) technologies typically focus their attention on end-user access; however, system-to-system integration is equally important and often considerably more complicated and costly. Fundamentally, internet computing is a platform that supports the open flow of information between systems. By leveraging ubiquitous technologies such as HyperText Transfer Protocol (HTTP), Hypertext Markup Language (HTML), Extensible Markup Language (XML), and JavaScript, that is a foundation for creating and transferring structurally complex documents across the Internet - the electronic (the Web-based) architecture delivers a set of server-based systems and services that support the true electronic systems integration.

A good example of the system electronization is the *PeopleSoft* Internet Architecture illustrated in Figure 5-1. It is a Four-Tier Architecture composed of the following levels:

1. Client Access – it can be a Web browser, a cell phone, or any external system,

2. Web Server – composed of the following services:
   - Portal Servlet[1] – based on JavaServlet that handles all inbound markup language and outbound requests for the Portal. It also manages all aspects of the *PeopleSoft* Portal such as search, content management, and home page personalization. It communicates with this back-end service via BEA System's JOLT,
   - Presentation Relay Servlet – based on JavaServlet that handles all inbound and outbound HTTP requests for *PeopleSoft* transactions and queries. This very thin servlet acts as a relay between the client device and the core back-end services. It receives and serves HTML, XML, and WML (wireless) requests over HTTP and maps the data in these requests to the Component Processor and Query Processor application services that execute under Tuxedo (Trans-

Figure 5-1: The PeopleSoft Internet Architecture



action Processing Monitor). It communicates with this back-end service via BEA System's JOLT,

- Integration Relay Servlet – based on JavaServlet that handles all inbound and outbound HTTP/XML requests for the third party system integration. This is also a very thin servlet that acts as a relay between the external or third-party system and the core back-end integration services. It receives and serves XML requests over HTTP and maps the data in these requests to the integration services – Application Messaging Processor, Business Interlink Processor, Component Processor – that execute under Tuxedo. This component communicates with this back-end service via BEA System's JOLT.

3. Application Server – applies Tuxedo – Transaction Processing Monitor (delivered by BEA Systems) to manage the following services:
   - Portal Processor – provides personalized and secured access to any system on the enterprise's internal and external network,
   - User Interface Generator – it dynamically generates the user interface based on the Component or Query definition and generates the appropriate markup language (HTML, XML, or WML) and scripting language (JavaScript, WMLScript) based on the client accessing the application,
   - Application Messaging Processor – manages the publishing, subscribing, and delivery of Application messages in a *PeopleSoft* system environment,
   - Business Interlink Processor – manages the execution of Business Interlink Plug-ins and their interaction with third-party systems, which for example have been acquired after the merger or in B2B inter-operations involving different platforms, such as UNIX and Windows NT,
   - Component Processor – is a key part of the Application Server; this processor executes *PeopleSoft*'s business application components,
   - Process Scheduler – executes reports and batch processes and registers the reports in the Portal's Content Registry,
   - Query Processor – executes queries defined using the *PeopleSoft* Query Tool,
   - Application Engine – Executes *PeopleSoft* application processes such as billing, loan handling, etc.,

- Security Manager – interfaces with the Enterprise Directory Server using Lightweight Directory Access Protocol (LDAP) to authenticate end users and manage their system access privileges,
- SQL Access Manager – manages all interactions with the Relational Database Management System (RDBMS) on its server.

4. RDBMS Server is the repository for all information managed by *PeopleSoft*'s enterprise applications. All *PeopleSoft* internet applications support industry-leading database management systems, including Oracle, Informix, IBM DB2, Sybase, and Microsoft SQL Server. Not only are application data stored in the database, but *PeopleSoft* metadata are also maintained in the database. The *PeopleSoft* Tools Application Designer development tools maintain this metadata which is then used to drive the runtime architecture. The *PeopleSoft* Application Server executes business applications based on this *PeopleSoft* metadata. Examples of common application objects are Fields, Records, Pages, Components, Application Messages, and Business Interlinks. When an application developer saves an application object, the Application Designer saves this definition to the PeopleTools metadata Repository. At execution time, the Application Server fetches the most recent application object definition from the Metadata Repository. It then compiles and caches in memory the application object definition and then executes the business application based on the definition. For example, when a Home Page is executed by the Application Server, the metadata definition is fetched, compiled, and cached. The page layout is generated based on this definition.

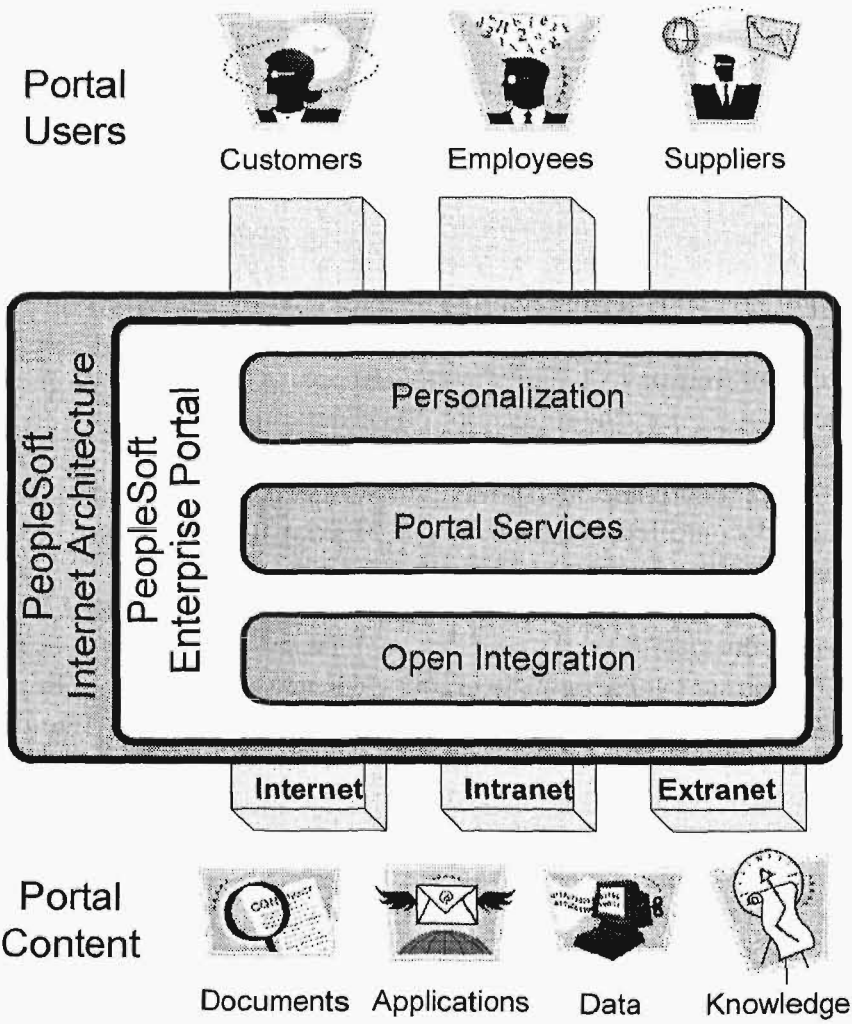
It is worthy to notice that using *PeopleSoft* Component Interfaces, third-party systems can synchronously invoke *PeopleSoft* business applications using COM, CORBA, EJB, or XML bindings, which will be described in the following sections.

PeopleSoft Portal is an important systems integration technology since it ties together content from a wide variety of data sources and delivers this content to end-users in the central User Interface Generator. Figure 5-2 depicts the architecture of *PeopleSoft*'s Portal. Its functions include:

- Personalized Homepage,
- Content Management,

- WebPublishing,
- Search,
- Workflow,
- Rule-based Access and Security,

Figure 5-2: The Architecture of PeopleSoft Portal



- XML Integration,
- LDAP Interface,
- Register new content,
- Upgrading,
- Metadata-driving,
- Multi-lingual,
- Multi-currency,
- Other.

SYSTEMS STANDARDIZATION

Distributing an application within a networked enterprise is not an end in itself. Distributed applications introduce a whole new kind of design and deployment issues. For this added complexity to be worthwhile, there has to be a significant payback. Some applications are inherently distributed: multi-user games, chat, and teleconferencing applications are examples of such applications. For these, the benefits of a robust infrastructure for distributed computing are obvious.

Many other applications are also distributed, in the sense that they have at least two components running on different machines. But because these applications were not designed to be distributed, they are limited in scalability and ease of deployment. Any kind of workflow or groupware application, most client/server applications, and even some desktop productivity applications essentially control the way their users communicate and cooperate. Thinking of these applications as distributed applications and running the right components in the right places benefits the user and optimizes the use of network and computer resources. The application designed with distribution in mind can accommodate different clients with different capabilities by running components on the client side when possible and running them on the server side when necessary.

Designing applications for distribution gives the system manager a great deal of flexibility in deployment. Distributed applications are also much more scalable than their monolithic counterparts. If all the logic of a complex application is contained in a single module, there is only one way to increase the throughput without tuning the application itself: faster hardware. Today's servers and operating systems scale very well but it is often cheaper to buy another identical machine than to upgrade to a server that is twice as fast. With a properly designed distributed application, a single server can start out running all the components. When the load increases, some of the components can be deployed to additional lower-cost machines.

With the advent of the Java programming language and the growth of the Internet, information technology (IT) managers are again excited at the prospect of using component software technology—the idea of breaking large, complex software applications into a series of prebuilt and easily developed, understood, and changed software modules called components—to deliver software solutions much more quickly and at a lower cost.

A component architecture for building software applications will enable the software provider to achieve economies of scale for software deployment by:

- **Speeding development**—enabling programmers to build solutions faster by assembling software from prebuilt parts.

- **Lowering integration costs**—providing a common set of interfaces for software programs from different vendors means less custom work is required to integrate components into complete solutions.

- **Improving deployment flexibility**—making it easier to customize a software solution for different areas of a company by simply changing some of the components in the application.

- **Lowering maintenance costs**—isolating software function into discrete components provides a low-cost, efficient mechanism to upgrade a component without having to retrofit the entire application.

A key goal of any component software architecture is to separate business logic—how a tax component calculates tax rates—from execution logic—whether the tax component runs in a browser or on a multiprocessor server. The following standards extend this separation even further because the same

components can communicate with each other across processes in a single computer or between computers over the Internet.

However, components by themselves do not solve all of the issues of enterprise application complexity. For example, suppose a business wants to rapidly build and deploy a customer order entry application that involves five different areas of functionality: tax calculation, customer credit verification, inventory management, warranty update, and order entry. The application will be built from five separate components and will operate on a Web server. How does the developer handle exceptions? System failures? Network outages? Peaks in performance load? Must these be hand-coded into the application? It defeats the two main goals of component-based development—fast time to market and lower development costs—if companies are forced to hand-code the mission-critical services that are required for online production systems.

To address enterprise requirements for a distributed component architecture without sacrificing rapid development and cost effectiveness, the following standardized architectures support this requirement.

Hundreds of applications and thousands of their objects (components) are distributed through the e-enterprise environment. To facilitate this distribution, particularly among objects (components) from software developed by different programmers and vendors, standards have been offered by some developers, such as OMG (CORBA), Sun Microsystems (EJB), Microsoft (COM), and others.

CORBA STANDARD

CORBA, which stands for Common Object Request Broker Architecture, is an industry standard developed by the OMG (Object Management Architecture Guide, a consortium of about 800 companies organized in 1989). CORBA is open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. Some large companies are embedding CORBA in networked devices for finance and medical applications.

CORBA is useful in many situations. Because of the easy way that CORBA integrates machines from so many vendors, with sizes ranging from