Web Applications Development Using J2EE Architecture

Stephen J.H. Yang, Canoe Hsih, Sandy Huang, and Jeff Huang
Department of Computer and Communication Engineering
National Kaohsiung First University of Science and Technology
jhyang@ccms.nkfust.edu.tw

Abstract

Through Internet, it is more complex to develop Web-based application including all various ways of business logic on World Wide Web. It is important that development time and economy cost are key values of customer application to the enterprise. We will explain the advantage about J2EE, and show how to use J2EE's architecture to build complex Web applications. Step by step, we bring up probable Web architectures including a flow for designing enterprise Web-based applications. We introduce J2EE's Enterprise Java components including Servlet, JSP, and enterprise beans and comply J2EE' blueprints to construct not only simply but also strongly functional dynamic pages contained dynamic content which would be deployed inside the presentation tier of Web applications and build reusable, manageable, maintainable, and high-performance enterprise business objects which would be deployed inside the middle-tier of its Web applications. And the UML that published by OMG noncommercial organization is most populating model language. So in this paper, we describe our system by UML.

We propose building business objects with EJB-compliant to encapsulate business logic with synchronous and asynchronous operations. Enterprise business is multifaceted, but it always encompasses e-commerce (B2C) and business-to-business solution. So we implement and illustrate a simply Web-Store containing B2C and B2B which will be applied our technique and J2EE's technology.

Key Word: UML, J2EE, EJB, Servlet, JSP, Container, Web Applications, JMS

1. Introduction

Now the master stream of the enterprise business building is tending to provide Web-based service. For the service, it point that enterprise's business go on network. Web-based applications require database connection, user authentication, session management and dynamic Html generation. Building the Web application for Web-based service is composed of the service of Web server, application server, and database server. More recently, application servers have enabled the aggregation of enterprise services on the back-end into a Web-facing display. Application servers abstract some of the complexity of enterprise application programming by implementing middle-tier services. The application server software provides a key component for companies that want to make their systems accessible for Internet-based e-commerce applications.

To build large, multi-tire, web-based applications require a large number of software service, from dynamic HTML generation to business domain object persistence. Rather than implement these services from scratch, application developers find it more efficient to use standard, publicly available implementations. Java programming model based on servlets, JSP and Java bean provides a standard, open, and robust Web architecture [3,4]. Java 2 Platform, Enterprise Edition (J2EE platform) is released from Sun Microsystems [1,3,4,7,16]. J2EE lets application developers build software by concentrate service implementation, which provided by software vendors. When Sun released J2EE to capture the growing e-business market, it changed Java from a language to an enterprise platform. The J2EE's goal defines a standard of functionality that help these meeting the challenges and thus increases the competitiveness of enterprise in the information economy and e-business. The J2EE platform supports distributed applications that take advantage of a wide range of new and evolving technologies, which simplifying development through a component-based application model. The J2EE platform makes all Java enterprise APIs and functionality available and accessible in a well- integrated fashion, simplifying complex problems in the development, deployment, and management of multi-tier server-centric enterprise solutions. So the J2EE model supports applications ranging from traditional client-server applications delivered over corporate intranets to e-business Web applications on the Internet.

The J2EE platform offers several benefits: (1) Simplified architecture and development. (2) Scalability to meet demand variations. (3) Integration with existing information systems. (4) Choices of servers, tools, components. (5) Flexible security model. Through the J2EE benefit, enterprise will provide their online business service inside their Web-based application with server-side components implementing enterprises' business logic. The J2EE platform is a scalable platform that can adjust to volatility of Web traffic. It can connect to virtually the enterprise's existing database or legacy system and allows enterprise managers to develop and deploy best-of-breed solutions based on products and technologies from multiple companies. J2EE is technology that enables software developers to write the guts of business components and applications such as connections to databases and handling transactions that will run across different computing systems.

Because the business's work is more and more complex, so the business web-application is increasing popularly and going to enormous. When the complex business web-application is increasing, the design, develop, and maintain of complex web application is becoming default and complex. In order to solve the problems, we can abstract and model system to manage the complexity. Today UML is most popular model language, which is established by OMG that is a noncommercial organization. However there are have more and more system are being described in UML notation [13,14,20]. Along the trend, we will use UML to modeling our system in this paper.

The structure of this paper is as follows. Section 2 discusses the benefit and challenge with Web and n-tier architecture. Section 3 introduces why J2EE, J2EE blueprints, Java Message Service [6,1215], and J2EE server-side components. Section 4 makes contrast and suggests some design issue as the guidance of J2EE components' usage. Section 5 design and implement a case study about Web applications of B2B and B2C applying our argument and J2EE's technology on J2EE platform. Section 6 concludes this paper, and discusses the future work, which combines other technologies with J2EE on developing Web-based applications.

2. **Related Work**

Web-based application development is very popular and fast. Its architecture can be three cases from past to recent development; basic Web site architecture, dynamic Web site architecture, and complex Web applications architecture.

2.1 Basic Web Architecture

About a basic Web site architecture, it includes three principal components including a Web server, a network connection and one or more client browsers. This is simplest client/server architecture on the Internet. As Figure 1, Client browsers request the Web server by making over a HTTP connection and using the HTTP protocol. Clients request Web pages with providing specific path information (URL) and pages' name. Web Pages' Content always was stored simply with formatted HTML tag and static content. When clients browse same Web pages, they always see the same content every time.

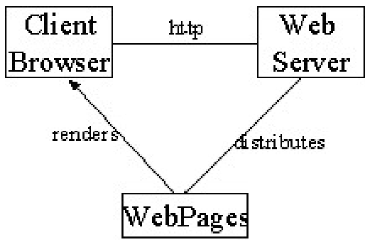


Figure 1. Basic Web Architecture

2.2 Dynamic Web Architecture

But information always was changed so quickly, that the content of Web sites must be updated to go with the stream. Then the contents of Web pages cannot be stored inside the Web pages, instead of assembling them at runtime from information which will be stored in a database, other information repository or formatting instructions in a file. Alternatively it can come form the output of a load-able module (CGI or ISAPI). The Web server interprets and executes the scripts in the page with using a page filter. As the Figure 2, it is simplest dynamic Web site Architecture. When a HTTP request is sent from a browser to the Web server, an external script is run on the server so that the Scripted Page can be processed to create the Web page with built-in APIs through referring back-end database to creating dynamic content.

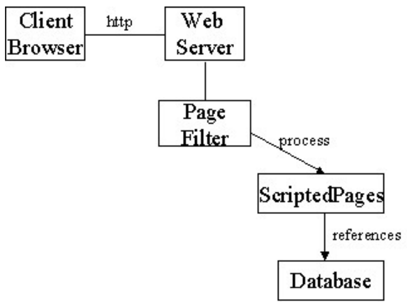


Figure 2. Dynamic Site Architecture

2.3 Complex Web Architecture

The development of Internet and World Wide Web technology results in the arrival of e-commerce [8,17], and has significance impact on business, commerce, industry, bank and finance, education and our personal and working life etc. Now recent Web application, it is n-tier architecture and also including complex business logic inside. In back-end, it contains business objects performing complex business logic in the middle tier, in addition to migrate legacy information and database systems to the Web architecture. As the Figure 3, at the runtime, the Web server calls the interface of the business objects, which has access to all the information that came along with the page request. The business objects use the request details, and typically accesses server side resources to produce the HTML stream with the combination of data and page templates (i.e. HTML template, CSS) returned to the client.

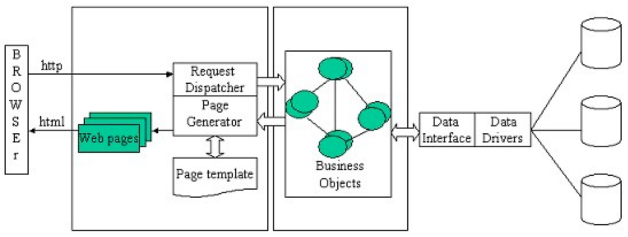


Figure 3. Complex Web Architecture

3. J2EE architecture and components

3.1 Why J2EE

The J2EE enables a component-based approach to development enterprise application. So J2EE reduces the cost and complexity of developing large, multi-tire web-based application. J2EE has two benefits. First, the key business benefits include: Industry buy-in and choice of vendors, faster time to market, greater utilization of existing IT assets, greater protection of future IT assets, a faster response to change, and choice of proven, scalable, and reliable platform. Another is the technical benefit including: component types supported, state management, database caching, declarative (automated) persistence, and scalability through load balancing.

3.2 J2EE Blueprints

J2EE architecture is composed by many software components [11,16]. It is based on servlet, JSP and Java bean technologies [9,10,11,18]. Java servlet and Java Server Page is used to communicate with user and present system response in browser. Java bean is to encapsulate business logic and can be accessed by client program. Based on the section 3.1 description of J2EE benefit, we know the advantage of using J2EE to develop. However how do we construct enterprise Web applications with J2EE technology? First, We need to know the overview and practical applications of J2EE platform, so we will explain about J2EE Blueprint including J2EE Environment, J2EE Containers, and J2EE Services.

The J2EE platform provides application designer with multi-tier models and container-based component management. Figure 4 illustrates the environment in J2EE platform. About multi-tier model, J2EE architecture defines three tiers including client tier, middle tier (consisting of one or more sub-tiers) and enterprise information systems tier (including enterprise information stored in RDBMS and integrating with enterprise legacy systems). About container-based component management, J2EE components live inside J2EE containers, that is, containers are standardized runtime environments that provide specific component services.

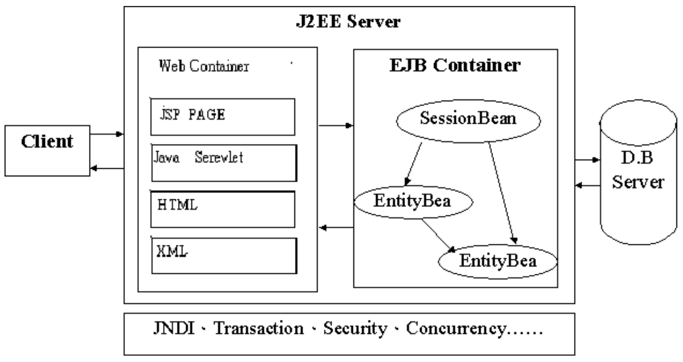


Figure 4. J2EE Environment

The multi-tier architecture of the J2EE platform is based on J2SE platform. And all components operate and communicate with each other inside J2EE containers using J2EE API and services defined in J2EE specification.

The client tier supports both applet container and application client container. In applet container, client tier can support more complex user interaction through Web browsers. That is, client-tier Java beans components would typically be provided by the service as an applet that is downloaded automatically from Internet into a user's browser. Another, client-tier bean can be a stand-alone application client stored in application client container. That is, enterprise would typically install a client program written in the Java programming language.

The middle tier that would be implemented presentation and business logic inside consists of both Web container and EJB container. In Web container, JavaServer pages (JSP), and Servlet are main non-pure Java components mixed with plain static pages [11]. They can also generate dynamic HTML with dynamical contents coming from back-end tier and interact with client tier components. In EJB container, business logic is implemented as Enterprise JavaBean (EJB) components (also referred to as enterprise beans). The EJB components model is the backbone of the J2EE programming model.

The J2EE platform, together with J2SE platform, contains a number of industry standard APIs for access to existing enterprise information system and database. They include the following services: :(1) HTTP. (2) HTTPS. (3) Java Transaction API (JTA). (4) JavaIDL. (5) Java Naming and Directory Interface (JNDI). (6) JavaMail and JavaBeans Activation Framework. (7) RMI-IIOP. (8) Java Message Service (JMS)

3.3 Server-side components about J2EE

The J2EE's server-side component is based on three Java programming technologies: Servlet, Java Server Page, Java Bean or Enterprise Java Bean [2,5,19].

3.3.1 Servlet/JSP Component In Web Container

A Web container as well as a JSP/Servlet container is runtime environment where Servlets ans JSPs live, it acts as a bridge between the client and EJB containers. A servlet is a java plug-in with web browser. The Java Server Page is a simple but power technology to dynamically generate HTML web page on the server side. In J2EE platform the presentation logic between end user and system response should be handled by Servlet and Java Server Page technologies, and the business logic should be encapsulated by Java Bean(or Enterprise Java Bean).

When an HTTP request with a Servlet/JSP comes from a browser through HTTP protocol, the Web server delegates the request to the container. For the first HTTP request, one application object (the ServletContext object) is created to provide application-wide services for all Servlet/JSPs within that Web application; for each client (browser), one session object (one HttpSession object) is created to provide services to the corresponding browser connection and be responsible for maintaining individual browser information; And for each HTTP request, a new request object (one HttpRequest object) and a new response object (one HttpResponse object) is created. The request object provides browser request information to a Servlet/JSP and provides data including parameter names and values that were set in the browser. A response object will be used by the Servlet/JSP container sending responses back to the browser. Another important concept of the Servlet/JSP container is that each HTTP request is handled in a separate thread within the Servlet/JSP container.

Now we know the operation and the HTTP request process of a Servlet/JSP container. But What is JSPs different from Servlets? Servlets are platform-independent Java server-side modules that fit seamlessly into the framework of an application server. Because Servlets are Java bytecode that can be download or shipped across the network, they are truly “Write Once, Run AnyWhere.” JSP is the Java platform technology for building applications containing dynamic Web content as HTML, DHTML. A JSP page is a text-based document containing static HTML and dynamic action that describe how to process a response to the client. The Servlet container has one Servlet engine which enables the dynamic content from Servlets to be displayed in HTML form your Internet application needs. And a JSP container provides the same services as a Serlvet container and JSP engine that interprets and processes a JSP page into a Servlet.

3.3.2 EJB component in EJB container

Enterprise Java Beans (EJB) is a technology in J2EE platform, which standardizes how to build server-side components for distributed applications. It is a reusable software component, which is written by java programming language. Then EJB components run within an EJB container, which will controls the beans and manage the execution of components with providing system services. The container is a run time environment that controls the EJB and provides them with some importance system-level service. So you don’t have to development these services yourself. In other words, EJB container is an interface between enterprise bean and EJB server. When client sends a request with JSP, the JSP file which in server side requests information from EJB. Once EJB generates content from database, then JSP file can display content to client.

There are four types of enterprise Java beans components in EJB2.0 specification that are: (1). ***Stateless session beans:*** Provide a service without storing a conversation state between methods calls; they are anonymous in that they contain no user-specific data. (2). ***Stateful session bean:*** Contains conversation state on behalf of the client; each instance is associated with a particular client.(3). ***Entity beans:*** Represent an object view of persistent data, usually rows in a database. They have a primary key as a unique identifier. Two operational styles for entity beans: container-managed persistent (CMP) and bean-managed persistence (BMP). (4). ***Message-driven beans:*** A newly added characteristic in EJB 2.0. These beans integrate between JMS (Java Message Service) and EJB to be used to perform asynchronous works.

A session bean represents a client, which want accesses service by invoking the session bean’s method in the J2EE server. So a session bean encapsulates business complex logic to reduce client’s workload. An entity bean represents an entity, which in real world that kept persistent storage machine like database. In conclusion, session bean is focus on business logic flow. Entity bean is focus on to represents data which in database. But when the business logic flow that encapsulated by session bean needs some data, which in database, session bean can accesses the data in database via entity bean.

In general, the J2EE industry products always support session beans and entity beans. Each bean must have itself home interface and remote interface. When they are deployed in an EJB container of the EJB Server, the home object of the EJB component will be created for implementing the bean's home interface. So a client can obtains a reference to the EJB by looking up the bean's registered nametag in the JNDI naming Service. The naming service returns a reference to the home object. The client then obtains a reference to a new or existing bean by invoking a create method or finder method on the home interface, respectively. For each client, the EJB container will create a remote object called an EJBObject to implement the beans' remote interface, which contains all the business methods that the bean exports. Then client could use remote object's methods to perform the same as methods of the bean instance.

In EJB Architecture, EJB containers' responsibilities are that: (1). Implement a home interface to an EJBHome object that allow clients to create, find (only for entity bean), and remove an EJB object. (2). Implement remote interfaces to EJBObjects objects that allow clients to execute the bean instance's methods. (3). Maintain EJB's naming, transaction, security, and persistence services. (4). Maintain EJB's state and lifecycle (creating, registering, and removing the bean).

Entity Bean: Entity bean represents data, which in database by object format. So entity bean is an object view of related records in the database or an existing application. Figure 5: represents the life cycle of an Entity Bean.

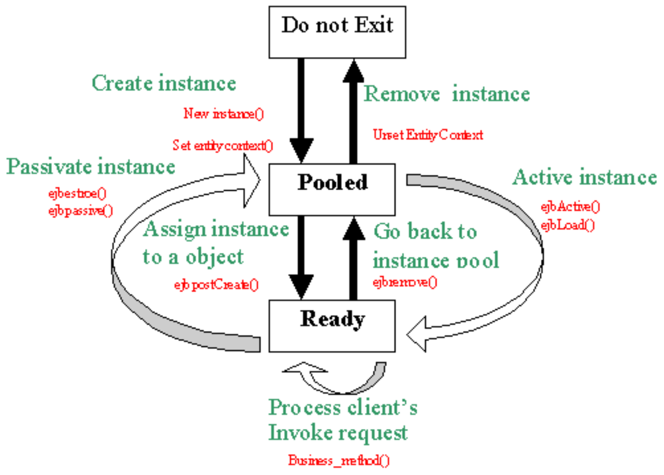


Figure 5. The life cycle of an Entity Bean

An entity bean instance is in one of these three states: *Does Not Exist*, *Pooled*, and *Ready*. When the entity bean instance is not created or in the Pooled state invokes useEntityContxt method, it is in the Does Not Exists state. When entity bean instance is created but not assigned to any EJB object, it is in the Pooled state. So all instance in Pooled state are equivalent. The Ready state means that when entity bean's lift cycle is controlled by EJB container from create bean instance, assign bean instance to EJB object, change bean's state and remove bean instance.

When the EJB server is started, it instantiates several instances of the bean which are placed in the *Pooled* state and has no arguments through invoking the new instance method.

This means that the instances themselves do not represent any data in the database. In the instance pool, a bean instance is available to the container as a candidate for serving client requests. All instances in the *Pooled* state are equivalent. A bean instance, which will be available to accept client requests, moves to the *Ready* state, when the container assigns it to an EJB object. And the bean will also be associated with data referred to one or more record in the database. This occurs under two circumstances: When a new entity bean is being created or when the container is activating an entity.

Session Bean: The EJB container is the runtime environment for Session Bean and Entity Bean. But something is different from those beans managed inside it. The Session Bean has stateful and stateless type. About the stateless session bean, the container maintains its lifecycle as shown in the Figure 6. Inside all lifecycle figures in our thesis, the blue line indicates the operation controlled by EJB container, and the brown line indicates the operation controlled by client.

The stateless session bean's life cycle has two states: *Does Not Exist* state and the *Ready Pool.* When a bean instance is in the *Does Not Exist*, it is not an instance in the memory of the system. When it enter the other state as the container needs them. So beans' state transitioning (lifecycle) is controlled by its container invoking ejbcreate and ejbremove methods of the bean. A stateless session bean does not maintain informally state for a particular client. Except during bean's method invocation, all stateless session bean instances are equivalent that allow EJB container to assign an instance to client. When a client invokes a business method on an EJBObject of a stateless Session Bean instance, the method call is delegated to any available instance in the *Ready Pool.* While the instance is executing the request, it is unavailable for use by other EJB objects. Once the instance has finished, it is immediately available to any EJB object that needs it. So that is why a stateless Session Bean instance provides a service without storing a conversation state between methods calls for clients. In addition to these, stateless Session Bean instances have the same identity when they have the same home object.

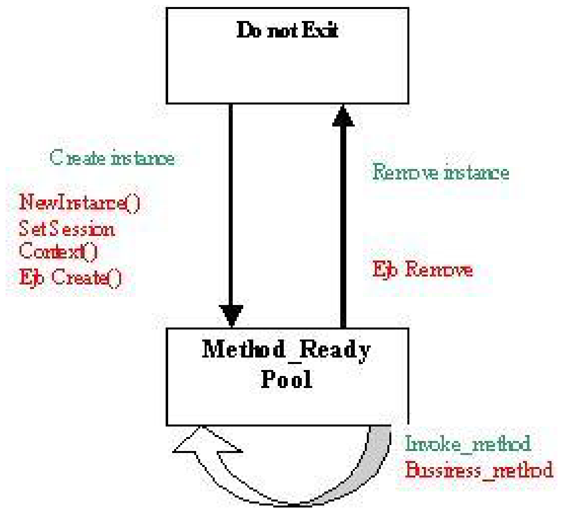


Figure 6. The life cycle of a stateless Session Bean

Like the stateless Session Bean, the stateful Session Beans has the third state called *Passive*, in addition to these two states. The different is that Stateful session beans are dedicated to one client for their entire life, so the client can control the ejbcreate to create the bean instance, and the ejbremove to remove the instance through invoking on an EJB home of a stateful session bean. To conserve resources, the container can passivity the bean instance which is not servicing methods from the client, while it is inactive by preserving its conversational state and evicting the bean instance from memory to read and write it's fields to the secondary storage associated with the EJB object. At this time, the bean instance should close any open resources and set all no transient, nonserializable fields to null. When the client invoke the service of it again, then the container will activate it from *Passive* state to *Method-Ready Pool* state. Or the container can remove the bean if it times out. During the bean lifetime, its instance will be passivity and activated zero or more times. Not like the stateless Session Bean, each stateful Session Bean's instance is associated with a unique identity when the instance is created. So the stateful Session Bean can contains conversation state on behalf of the client.

There are the following session bean characteristics. First, a session bean needs to communicate with a client. Second, a session bean can act as mediator of communication between clients and EJB tier. Third, a session bean can encapsulate work flow specific to an application or serve as an entry point to a hierarchy of information keyed to an attribute of the entry-point bean.

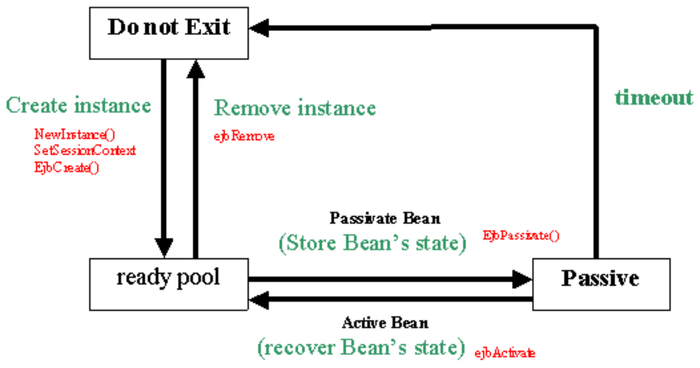


Figure 7. The life cycle of a stateful Session Bean

Message-Driven Bean: The release of EJB 2.0 specification resolved these challenges by extending the EJB component model to provide simplicity to bean developers needing support for asynchronous messaging -- the new message-driven bean type. Message-driven beans, the integration between JMS (Java Message Service) and EJB, are used to perform asynchronous with the server. Unlike entity bean and session bean, the message-driven bean can't invoke by client. The responsibility of message-driven bean is reacts to the event of message-queue. That is to say, when a message comes to message-queue, the message-driven bean will be to drive. So client does not directly invoke the message-driven bean, it is driven by message-queue. In a word, the entity and session bean is a synchronize model and the message-driven bean is an asynchronously model of EJB.

To a client, message-driven bean is a message consumer that implements business complex logic. The client sends messages to the destination or endpoint, and then the message-driven bean will handles the processing of the messages. There is no client view to a message-driven bean, meaning that there are no home and remote interfaces. The life cycle of message-driven beans is stateless. When a message producer (client) writes to a topic or queue, then the container delegates a received message either to an existing method-ready instance or a new instance allocated to handle the message. Likewise, similar to stateless session beans, message-driven beans are anonymous, having no identity to a client.

Why do we need use the message-driven bean to construct our component? Fist, message-driven beans provide a component model around JMS messaging and support messaging in component execution environment (e.g., Container), which is particularly important for scalability and portability. The Container promotes scalability through its efficient resource management while processing message. Concerning portability, it offers portable access to resource factories and resource environment entries. It links references to existing resources in the environment in a portable manner by using structural information form the bean's deployment descriptor. So with message-driven beans you don't have to develop special "start-up" classes that initialize JMS destinations at application server start-up. Second, Analogous to the state characteristics of stateless session, Message-driven beans have no conversation state. All bean instances are equivalent when they are not processing a message. It is natural way to manage message-driven bean instances efficiently by using the pooling of bean instances. Third, message-driven beans live by the threading restrictions imposed on all EJBs. This restriction eases development of message-driven beans because the container performs concurrency control for developers. Developers do not have to worry about developing thread-safe business logic.

4. Develop Complex Web application with J2EE components

Because Internet is going to power so business application is trend to web-based application. The J2EE is a component-based architecture to development distributed objects that using Java-based language. In this section we will discusses the deign issues of developing container-centric Web applications form Web container to EJB container.

4.1 Building Web Applications with Servlet

When programmers build simple Web-based applications, they can only write simple static pages to plain HTML pages and dynamic pages to Servlet. See Figure 8, a Web container contains these two components. The browser requests are delegated to plain HTML pages or the Servlet container. As descriptions of the 3.3.1 section, Servlets will take HTTP requests from a Web browser, generate the request dynamically and then send a response containing an HTML document to the browser. And Servlets also can connect to database system by themselves through JDBC service. One better choice is that Servlets import Java classes to connect back-end resource. About this architecture design issue, Servlet is an all-powerful role. Servlet responsibility includes presentation logic, business logic, and coordination operations mediating business methods calling and display pages presenting. Java classes can play business objects and resource-access roles.

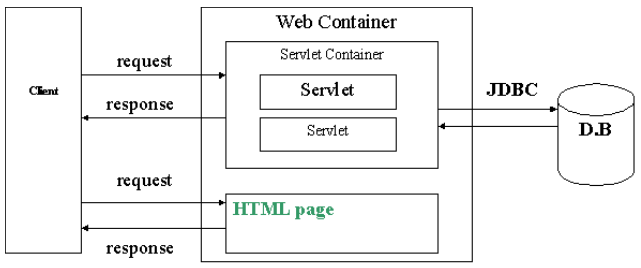


Figure 8. Architecture Design Issue with Servlet

What is the advantage or disadvantage of using this design issue? About advantage, Servlet gets dynamic content to present. A Servlet class processes clients' requests by the way being one thread for one HTTP request. About disadvantage, Java Servlet must handle the creation of HTML pages. Embed HTML presentation with Java code will cause high coupling between presentation logic and other logic including request-handling logic and business logic.

4.2 Building Web Application with JSP and Servlet

About this section, we promote the design issue of Servlets and JSP working together. Developers can choice JSP to serve dynamic Web content. Developers write a significant amount of scriptlets or Java code embedded within JSP pages. A JSP page can use Java beans (Java classes) through the standard tag library to action them. Also, JSPs are allowed to use customer tag libraries to produce customer actions. The JSP specification allows for including one JSP page with another. By using tag library and including methodology on JSPs, developers can encapsulate specific functionality of pages and increase reusability and maintainability. Java beans (Java classess) are responsible for business logic and resource accessing performed through some Java API services. As Figure 9, we indicate the architecture design issue.

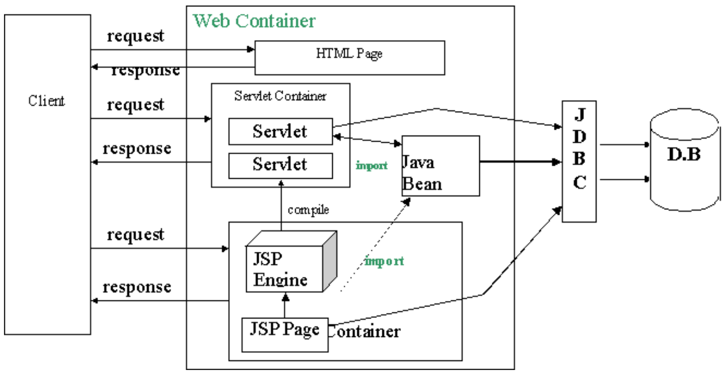


Figure 9. Architecture Design Issue with JSP and Servlet

It is important that how to cooperate with JSPs and Servlets and what jobs they work, when they are together. The Servlet acts as the mediator and is in charge of the request processing and the creation of any beans or objects used by the JSP, as well as deciding, depending on the user's actions, which JSP page to forward the request to. The JSP just plays a presentation role, which there is no processing logic within. It is simply responsible for retrieving any objects or beans that may have been previously created by the Servlet, and extracting the dynamic content from that Servlet for insertion within static templates. And Java classes are business objects performing business logic or business rules. In our opinion, this approach typically results in the cleanest separation of presentation from content and business logic, leading to clear delineation of the roles and responsibilities of the developers and page designers on your programming team.

4.3 Building Web Application with EJB

It seems enough to construct business objects with Java beans. But Developers feel difficult and helpless, while they construct business components including security, transaction, and persistence services. As the description of section 3.3.2, we know these machines and technology of EJB container and EJB. Construct EJB-centric Web application extends the modular, component-based application described in the previous section. Figure 11, it is our design architecture in which there is a bridge being EJB container between Web Container and database. Servlets, JSPs and Java beans can communicate synchronously with Session beans and Entity beans by locating these beans with JNDI naming service. Java bean and Session beans can message asynchronously to message-driven beans with messages encapsulating any Java data type. Session beans, Entity beans, and message-driven beans, which play business components, can retrieve and access back-end resource. EJB components are demanded easy to performing security, transaction, and persistence service to client by EJB containers' monitor and management.

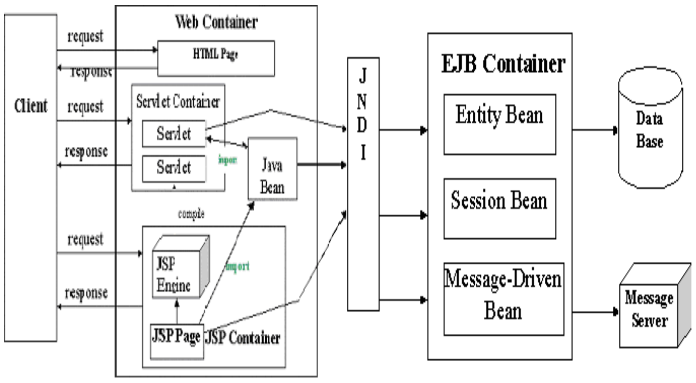


Figure 11. Architecture Design Issue with EJB

So in this architecture, the Web container hosts Web components that are almost exclusively dedicated to handling a given application's presentation logic. And the EJB container host's business components that, on the one hand, respond to requests from the Web tier, and on the other, access the EIS resource. The Servlet also plays a presentation coordinator, which decide what JSP to display by depending on the user's actions, and a request deliveryman, which is in charge of the user's requests delivery and the creation of necessary Java beans or classes to processing these requests. The Java bean is a request processor and a business coordinator, which will coordinator business method callings. The JSP still presides at the presentation as same as the role in the previous section. The EJB plays the business component that includes business methods implementing business rules.
About our design, the advantage includes:

(1) EJB container management: container-managed service, as transaction and persistence, can be assigned easily to an EJB bean or bean's methods.
(2) High efficiency and good reusability: enterprise business components are constructed to be suitable for four EJB types, that will increases higher efficiency and better reusability than pervious design.
(3) Separating stateful and stateless business processes: two styles of session beans can separate stateful and stateless business logic or process and represent work flow between business objects.
(4) Encapsulate persistent data to a data component: An entity bean is a business object that represents persistent data and provides concurrent access by multiple clients. So an entity bean combines the functionality of persistent data with the convenience of object encapsulation.

The above design issues provide a better direction to develop multitier Web-based application with EJB components implementing business logic. Some discussions are important to know about, how to construct EJB object to business objects and what relationship among them.

4. Implementation and Illustration

4.1 System environment and requirement

Our software architecture mainly contains: clients are generally web browsers like IE; our application server called AP server is to use the BEA WebLogic Server version 6.0 with Service Pack 1 which supports J2EE platform and EJB 2.0 specification; the database server is the Oracle RDBMS. The AP server uses the type 2 JDBC drivers called jDriver for Oracle to connect Oracle database. The database version must be Oracle version 8.1.5 or over the version which the AP server supports. Our system has two retailers system and one wholesaler system. One retailer system called RetailerA is a web store selling notebooks. And another is very simple java application called RetailerB
The system functionalities are composed with B2C and B2B. Figure 12 is the Use Case diagram of our system. At B2C, it is a Web store called a RetailerA system containing registering, login, and shopping services for customers. At B2B, There are purchasing and quotation between the retailer systems and a wholesaler system. We also draw ERD diagrams to help building tables of our implementation. As shown in Figure 13

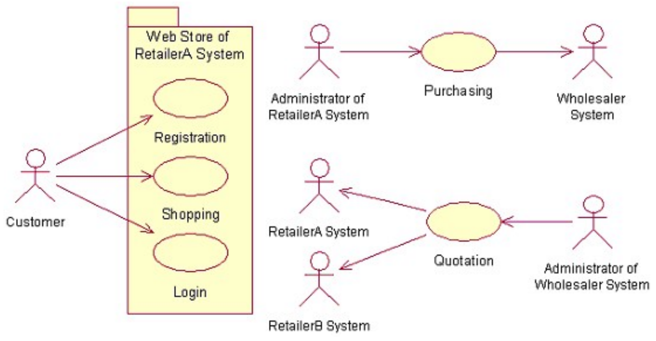
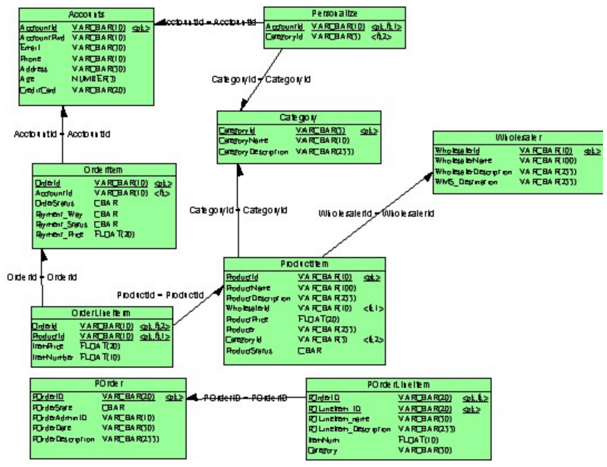


Figure 12: The Use Case Diagram of the B2B2C Implementation



Accounts
AccountId VARCHAR(10) <pk>
AccountPwd VARCHAR(10)
Email VARCHAR(30)
Phone VARCHAR(10)
Address VARCHAR(50)
Age NUMBER(3)
CreditCard VARCHAR(20)
AccountId = AccountId
Personalize
AccountId VARCHAR(10) <pk,fk1>
CategoryId VARCHAR(5) <fk2>
CategoryId = CategoryId
Category
CategoryId VARCHAR(5) <pk>
CategoryName VARCHAR(10)
Category Description VARCHAR(255)
Wholesaler
WholesalerId VARCHAR(10) <pk>
WholesalerName VARCHAR(100)
WholesalerDescription VARCHAR(255)
WMS_Description VARCHAR(255)
AccountId = AccountId
OrderItem
OrderId VARCHAR(10) <pk>
AccountId VARCHAR(10) <fk>
OrderStatus CHAR
Payment_Way CHAR
Payment_Status CHAR
Payment_Price FLOAT(20)
CategoryId = CategoryId
WholesalerId = WholesalerId
OrderId = OrderId
ProductId = ProductId
ProductItem
ProductId VARCHAR(10) <pk>
ProductName VARCHAR(100)
ProductDescription VARCHAR(255)
WholesalerId VARCHAR(10) <fk1>
ProductPrice FLOAT(20)
Producer VARCHAR(255)
CategoryId VARCHAR(5) <fk2>
ProductStatus CHAR
OrderLineItem
OrderId VARCHAR(10) <pk,fk2>
ProductId VARCHAR(10) <pk,fk1>
ItemPrice FLOAT(20)
ItemNumber FLOAT(10)
POrder
POrderID VARCHAR(20) <pk>
POrderState CHAR
POrderAdminID VARCHAR(10)
POrderDate VARCHAR(50)
POrderDescription VARCHAR(255)
POrderID = POrderID
POrderLineItem
POrderID VARCHAR(20) <pk,fk>
POLineItem_ID VARCHAR(20) <pk>
POLineItem_name VARCHAR(50)
POLineItem_Description VARCHAR(255)
ItemNum FLOAT(10)
Category VARCHAR(50)

Figure 13 The RetailerA ERD Diagram

4.2 B2C

Implementing the Login Use Case and starting which is the situation on starting the Web store, we use one Servlet, template JSP pages, two JavaBeans, one Session Bean and one

Entity Bean classes. The Figure 14 shows the class diagram.

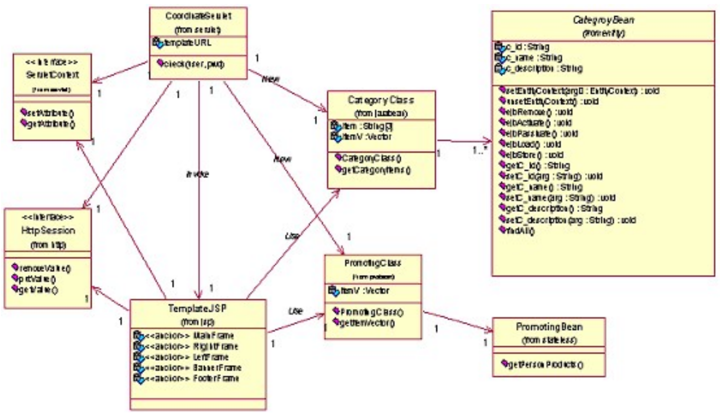


Figure 14 The Class Diagram of Starting and Login

There are application flows about Starting and Login. As the below Figure 15, there are six sequences to complete the flow of Starting. (1). Clients want to enter our Web store with a HTTP URL connection, which connects the CoordinateServlet class. (2). The CoordinateServlet news the CategoryClass JavaBeans. (3). When the CategoryClass is created; it will load and package the category information by locating all Category Entity Bean instances to capture category information. (4) The Servlet invokes TemplateJSP. (5). The TemplateJSP knows the client is a general customer. It uses the CategoryClass Java bean to get category information. Then it generated to be a general user's TemplateJSP responses to the client.

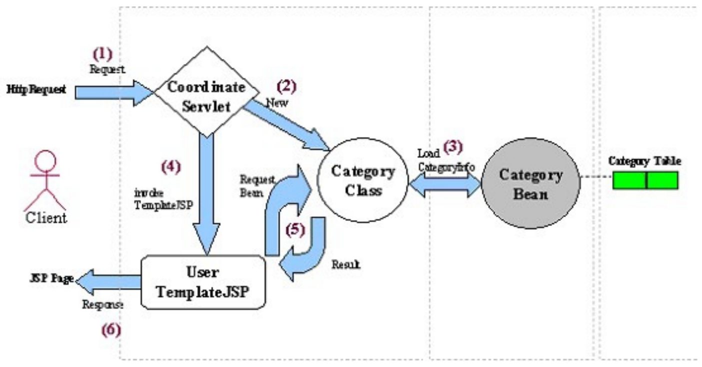


Figure 15 The Application flow of Starting

As shown in Figure 16, there also are six sequences to complete the flow of Login. (1). The client writes his information containing username and password in HTML page's input text fields and submits to the CooridnateServlet. (2). The CoordinateServlet gets his username and password to checks the client. If the checking is successful, the Servlet creates the PromotingClass with the client's information. (3). The PromotingClass gets the client's information to create the PromotingBean. It invokes the method in the PromotingBeam to get products' information (4). The CoordinateServlet invokes the TemplateJSP. (5). The TemplateJSP knows the client has an account. It uses the PromotingClass and the CategoryClass Java beans to get values. Then it generated to be the account's TemplateJSP responses to the client. Figure 17 shows the class diagram of the Registration with the RegisterResultJSP. The class diagram represents the relationship of correlative components processing client's register data.

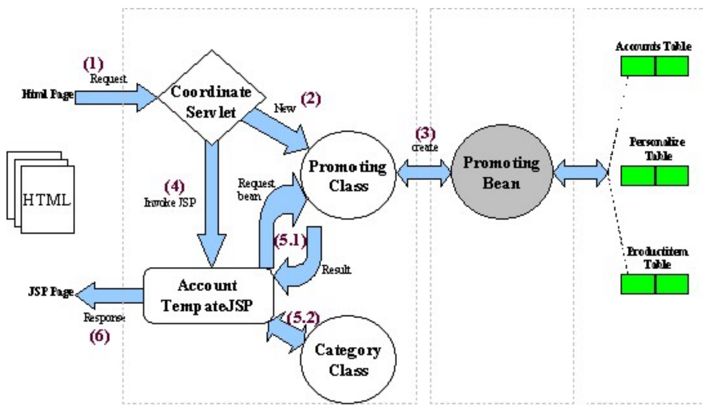


Figure 16. The Application flow of Login

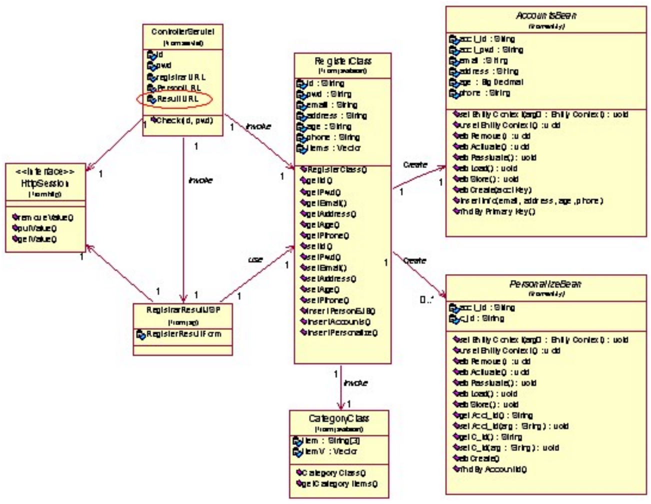


Figure 17. The Class Diagram of Registration with RegisteResultJSP

As shown in the Figure 18, there are seven sequences to complete the flow of processing client's register data. (1). A client submits the request, which is processing his register data, to the ControllerServlet. (2). The Servlet delivers the request to the RegisterClass. (3). The RegisterClass gets category information from the CategoryClass. (4). The RegisterClass saves client's data through creating entity beans. (5). The ControllerServlet invokes RegisterResultJSP. (6)(7). The RegisterResultJSP gets result information to response the client.

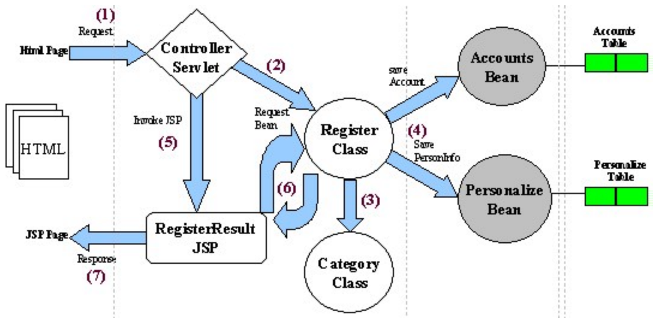


Figure 18. The Application flow of processing client's register data

A shopping cart will be created, while an account chooses products to submit at first time. The creation of a shopping cart is not described in here. Figure19 only shows the class diagram of the Shopping with ShoppingCartJSP. The class diagram represents the relationship of correlative components processing the behavior including add, update, delete, clear or remove operations on manipulations of shopping cart.

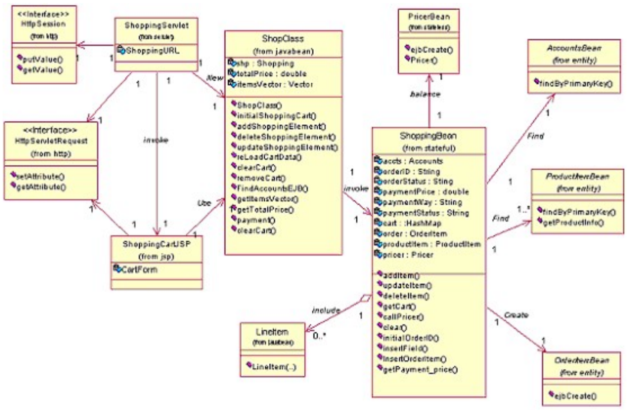


Figure 19. The Class diagram of Shopping with ShoppingCartJSP

Figure 20 shows the flow of manipulating the shopping cart. The sequence is: (1). A client chooses functionalities of the HTML shopping pages including add, delete, update, and payment function keys to submit. They submit the request with product items information to the ShoppingServlet. (2). The ShoppingServlet creates a new ShoppinClass instance and delivers entirely the request to it. (3). The ShoppingClass processes the request and perform the operation of manipulating the client's shopping cart. And it also gets and packages the information of the shopping cart. (4). The SoppingServlet invokes the ShoppingCartJSP. (5)(6). The ShoppingCartJSP uses the ShoppingClass to response the result as an account order to the client.

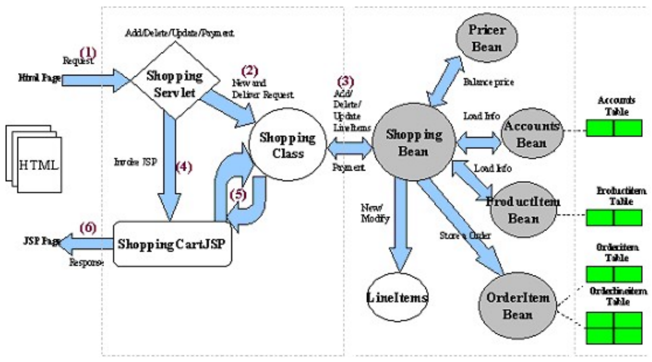


Figure 20. The Application flow of Shopping with ShoppingCartJSP

4.3 B2B

In the implementation of the Quotation UseCase, we designs mainly components including: QuotationServlet, QuotationClass, WquotationJSP, QuotationSessionBean, ProductState, QRetailerEvaluatorMDBean, ProductRule components. As shown in Figure 21, there are some sequences in the application flow of Quotation, including (1)~(5) in wholesaler system and [1]~[3] in retailer systems.

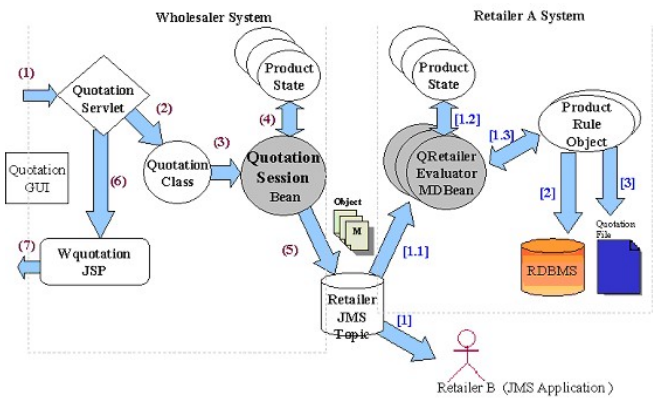


Figure 21. The Application flow of Quotation

In the implementation of Purchasing UseCase, we design mainly components including: PurchasingServlet, PurchasingJSP, PurchasingClass, PRetailerPuducerSessionBean, PWholesalerConsumerMDBean, WProductRuleObject, and PRetailerConsumerMDBean. As shown in Figure 22, there are some sequences in the application flow of Purchasing.

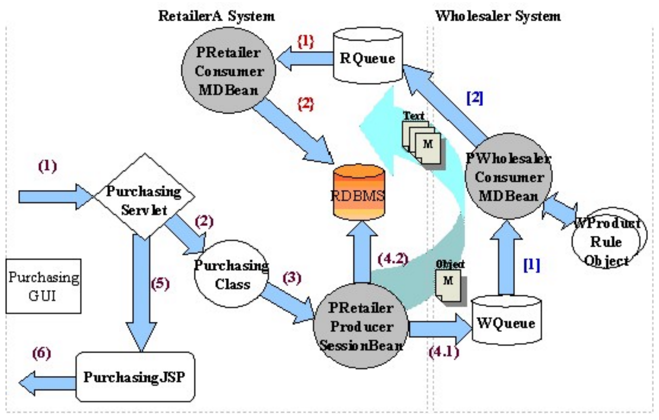


Figure 22. The Application flow of Purchasing

5. Conclusion and Future Work

In this paper, we propose some architectures and application flows of developing Web-based application on J2EE platform. Second, we also propose the design issue of building enterprises' business objects to J2EE's server-side components in object-oriented middleware. These design issues mainly are tends to make good use of and choose right type of EJB components of J2EE for help enterprise with constructing maturity and economy cost Web application. Our implementation and illustration are only just one simple example to present. In our approach, Enterprises can take easily their complex business to Internet and construct their Web-based service with integrating their enterprise information systems.

In the future, we will combine XML and J2EE technology to Web-based application, and discussion of asynchronously messaging, message-driven bean. While originally intended to act as an integration point for JMS, message-driven beans have the potential to become the industry standard for processing any asynchronous message. We will devote the development of XML-based messages and message-driven bean in future work. There are other communication mechanisms, as well as HTTP, FTP, and Email protocols etc, which may have been in place for a number of years in outside world. It is important that we use

JMS to communicate with theses outside entities that usually have established protocols with their electronic communication. As shown in Figure 23, we will devote building JMS clients as bridges or connectors to connect with different protocols.

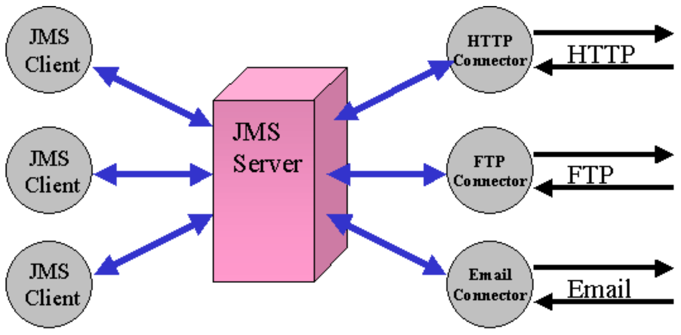


Figure 23. JMS Clients as protocol connectors to the outside world

Acknowledgement

This paper is supported by National Science Council in Taiwan under Grant NSC 90-2213-E-327-014.

References

1. N. Kassem and the Enterprise Team, "*Designing Enterprise Applications with the J2EE^{TM} 2 Platform, Enterprise Edition*", October, 2000, Sun Microsystems,Inc. http://java.sun.com.
2. Anne Thomas, "*Enterprise JavaBeans™ Technology: Server Component Model for Java™ Platform*", prepared for Sun Microsystems, Inc.
3. Sun Microsystems, "*Java™ Message Service Specification version 1.0.2*", JavaSoft, November, 1999, http://java.sun.com.
4. Sun Microsystems, "*The Java™ 2 Enterprise Edition Developer's Guide*", May, 2000, Sun Microsystems, Inc. http://java.sun.com.
5. Linda G. DeMichiel, Specfication Lead, L.Umit Yalcinalp, Sanjeev Krshnan, "*Enterprise JavaBeans™ Specification Version 2.0*", May, 2000, Sun Microsystems, Inc. http://java.sun.com.
6. Richard Monson-Haefel, David A. Chappell, *Java Message Service*, O'RELLY, 2000.
7. Nicholas Kassem, the Enterprise Team, "*Designing Enterprise Applications with the Java^{TM} 2 Platform, Enterprise Edition*", Version 1.0.1, Final Release, October, 2000, http://java.sun.com.
8. GemStone Systems, "*iCommerce Design Issues and Solutions Version 1.0*", February, 2000, Gemstone Systems, Inc. Beaverton, Oregon.
9. Eduardo Pelegri-Llopart, "*Tag Libraries in JavaServer Pages*", January 2000, Sun Microsystems, Inc.
10. Sun Microsystems, "*JavaServer Pages(TM) Tutorial*", 2000, Sun Microsystems, Inc. http://java.sun.com.
11. O'Reilly Conference on Java, "*Architecting the Web Layer: Putting your JSP and Servlets together*",March 29th, 2000, Santa Clara, California.

12. Sun Microsystems, "*Java™ Message Service Tutorial 1.3 Beta Release*", 2001, Sun Microsystems, Inc. http://java.sun.com.
13. J. Conallen, "*Modeling Web Applications with UML*", White Paper Conallen Inc, June 1998. http://www.conallen.com/ModelingWeb Application.htm.
14. Jingfeng Li, Jian Chen*, and Ping Chen of Software Engineering Institute of Xidian University Xi'an P.R. China, "*Modeling Web Application Architecture with UML*", IEEE, 2000.
15. Cristina Belderrain, "*Message-Driven Beans and Encapsulated Business Rules*", Organic, Inc. Sao Paulo, Brazil.
16. Robert Ipsen, *Mastering Enterprise JavaBeans^{TM} and the Java^{TM} 2 Platform, Enterprise Edition*, published by John Wiley & Sons, Inc.
17. Craig McClanahan, Larry McCay, and Erik Bergenholtz, "Web Application Framework", volume: 6 issue:3, Java developer's journal, March, 2001. http://javadevelopersjournal.com.
18. David Lyons, "*Creating a JSP JavaBeans Framework*", volume:5 issue:2, Java developer's journal, February, 2000.
19. Jason westra, "*E-Business with EJBs*", volume:5 issue:2, Java developer's journal, February, 2000.
20. Martin flowler with kendall scott, *UML Distilled Second Edition*, an imprint of Addison Wesley Longman, Inc.