2001

SpiderNet: A multi-server code service model design for computational grid support.

Xiaohong. Yu
University of Windsor

SpiderNet----A Multi-Server Code Service Model Design for Computational Grid Support

by

Xiaohong Yu

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

0-612-67605-6

Abstract

One goal of a Computational Grid is to aggregate ensembles of shared, heterogeneous, and distributed resources to provide computational "power" [Foster00]. The increasing use of Web technology Internet and Intranet is making the Web an attractive framework for solving distributed applications, in particular, because the interface can be made platform independent. Several systems have been developed that make distributed computing available. However, these systems are either not web-based or not taking the enterprise Java technology [Chen97]. Typically, those systems are unnecessarily complicated and require long development.

This thesis is an approach to research and development on distributed, multi-server code service system model ---SpiderNet. While a service-based architecture for distributed computer systems is not new, a theory and practice approach on using EJB in multi-server environment is presented in this thesis report. It adapts enterprise Java and web application server technology trying to solve the fundamental problem of multi-server model: work balance for efficiency and fail over for availability.

This thesis also demonstrates that with using Web and enterprise Java development toolkit, it is possible to build applications using a graphical "drag-and-drop" and web-based interface, thereby speeding up the application development.

To My Wife and My Family

Acknowledgements

There is a beauty which God gives at birth, and which withers as a flower. And there is a beauty which God grants when by his grace men are born again. That kind of beauty never vanishes but blooms eternally. There are many people I would like to thank for their helps. Without their helps, it's impossible to finish this thesis.

Thanks go to a great supervisor, Dr. Robert D Kent. His comments, encouragement, and patience were invaluable to the completion of this thesis. I would also like to acknowledge c3.ca and HPC Windsor for use of resources and SHARCnet for the awarding of support of research during the summer of 2001.

Thanks go to Dr. Saba and Dr. Schlesinger, for their valuable suggestions. Thanks also go to Dr. Morrissey for her chairing my thesis defense. I'm thankful to my committee for being very accommodating. I would like to thank Mary, Paul and other people of the School of Computer Science for their helps.

I would like to thank my wife Xiuling and my daughter Helen, for their understanding, consistent support and encouragement. During my study period, there were so many days and nights that I couldn't stay with them when they need me so much. Special thanks to my parents and relatives for their unconditional love and silent support.

Table of Contents

Chapter 1 Introduction

This thesis is primarily concerned with using enterprise java technology for multi-server and web application model design.

1.1 Motivations

Today the Internet becomes a standard vehicle for information exchange and very attractive resources for distributed computing. As more computer involved in Internet with diversity in hardware and software. [Seige00, Schne97] networked computing environment is becoming more diverse. Large-scale distributed software projects have often been specialized for their domain due to the limitations of communication technology and lack of common middleware technology.

Today's enterprise exposes 6 challenges for application developers [Subra00]:

Responsiveness: High-paced. fast-changing information driven economy means that responding quickly to new information is critical in establishing and maintaining a competitive edge.

Programming Productivity: To develop and deploy applications as effectively and as quickly as possible is very important.

Reliability and Availability: To get system running continuously with some fault tolerance ability is critical to success.

Security: To implement an effective security model is necessary and becomes more and more difficult.

Scalability: To make the application grow and meet new demand both in its operation and user base is very important. since an application's potential user may be millions of individual users through the Internet. The scalability requires not only the ability to handle a large increase in the number of clients but also effective use of system resources.

Integration: Since there are many developed system and much of information exists as data in old and outdated information systems, it's necessary for the new applications to be able to integrate with the existing information system. The ability to combine old and new technologies is key to the success of enterprise application development. Industry experience shows that integrating these resources can take up to 50% of application development time [SUN10].

The use of structured project management is critical to the success of current software development projects. With the appearance of Giga-byte network and the development of component and framework technology, it's possible to develop a large, effective, multi-tier and distributed application on heterogeneous platforms and resources. In contrast to traditional 2-tier client-server systems, systems of 3 or more tiers are the current technology trend. With the development of middle tier and framework technology, the applications will share a common need to couple devices that have not traditionally been thought as part of the application. [Foster00].

The research and development of grid computation has been started a few years ago and several grid application systems have been successfully developed. However, most of the systems are based on traditional distributed technology that is complex and inflexible instead of simple, uniform platform. Valuable effort and time are wasted due to repeat development of the same works.

Component technology and Enterprise Java technology have developed rapidly in recent years. The research and development of B2B, B2C framework greatly affects the application development pattern and developing time [Butle00]. The framework not only make it possible to accomplish file transfer, but also to direct access to computer, software, data and other resources. All the resources can work as collaborative problem solving and resource sharing/brokering. We will discuss more details about framework and component technology in the following chapter.

To build a framework in large, it need vary kinds of support including OS and database management system providers, middleware and tool vendors, vertical market applications, component developers and research institutions. A robust and flexible platform needs to be implemented on the wide variety software and hardware. Data representation format, transfer protocols, and underlying infrastructure support need to be standardized with flexibility. This infrastructure should be composed by many flexible components.

The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multitier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized modular components, a set of services to those components, and the ability of handling many details of application behavior automatically, without complex programming. The portal, openness and standardize of J2EE platform make it an ideal platform for building large scale distributed system.

1.2 Objective

Multi-client and multi-server is a fundamental problem for building distributed infrastructure framework. Workload balance makes servers working corroboratively, thus improve the system's efficiency. Fail over makes the system more reliable and available.

Our object in this thesis is giving an approach on designing distributed, multi-server computing model. We wish to design and construct the appropriate software model to support the available of possible database tools, networked type configuration and protocol, support for independent code development carry out within the context of a local database and extended to permit sharing, modification and reversionning of software. A prototype of multi-server system for code service---SpiderNet has been designed and built. Using Web Application Server (WAS) as the middle tier, SpiderNet system has multiple servers. It provides registry, user search, user update, code retrieve, code generate and update for multiple users concurrently access. The system can charge users according to the usage of the codes. It has an on-line billing system. It will be posted as a Website and can be accessed by global users concurrently. As a multi-server system, it possesses workload balancing and fault tolerance. Based on this design model, security, B2B, B2C and more complicated distributed applications can be further developed.

In this thesis report, we give an approach of adapting enterprise Java and web server technology to provide code service for users. In the following chapters, we will discuss component technology and multi-server prototype design.

The objectives in this thesis can be summarized as the followings:

- To develop a new approach for multi-server model design.
- To develop a multi-server system that can handle workload balance and fail over.
- To simplify design and development by adapting EJB technology on J2EE platform.
- To develop a prototype to validate this approach.

1.3 Organization of the Thesis

The remaining part of this thesis is organized as the following:

In chapter 2, we will introduce the component technology. We will briefly introduce servlets, CORBA, JSP, EJB, Jini and JavaSpaces. These component technologies are corner stone of J2EE platform. We will also introduce XML technology. They are essential for our prototype design. In the final part of this chapter, we will do the comparison of these technologies.

In chapter 3, we will introduce platform and EJB technology. We will give more detail of the design, architecture and utilization of EJB. At the end of this chapter, we will briefly overview the research of CodeNet and some existing computational grid application systems.

In chapter 4, we will explain using EJB in multi-server design. How EJB can handle workload balance and fail over in distributed system. We will introduce some concepts such as Idempotent, Smart stub, etc. EJB future development will also be discussed.
In chapter 5, we will cover the prototype design and implementation. Although it is implemented on Windows platform, it can run on any platform with Java virtual machine.

This prototype is not complete, but it sufficiently demonstrates many concepts and our design ideas. It will be posted as a Web site with its homepage.

Finally in chapter 6, we will cover the conclusions and discuss the future works.

Chapter 2
Background: Distributed System and Component Technology

A distributed system is a collection of independent computers that appear to the users of the system as a single computer. Distributed data processing refers to a data processing arrangement in which the data is decentralized and scattered in various places [Singh99]. Hence, processing occurs in a number of distributed locations and only semi-processed information is communicated on data communication lines from remote points to the central computers. It requires two or more distinct processors to complete a single transaction, and the distribution of applications and business logic is spread across multiple processing platforms.

2.1 Distributed System Architecture and Characteristics in Heterogeneous Network

Figure 2.1 illustrates the client using a distributed object model. Client/server architecture is a versatile, message-based modular infrastructure intended to improve usability, flexibility, interoperability, and scalability as compared to a centralized, mainframe, time-sharing approach to computing [Siege00]. This model is typically used in traditional distributed programming.

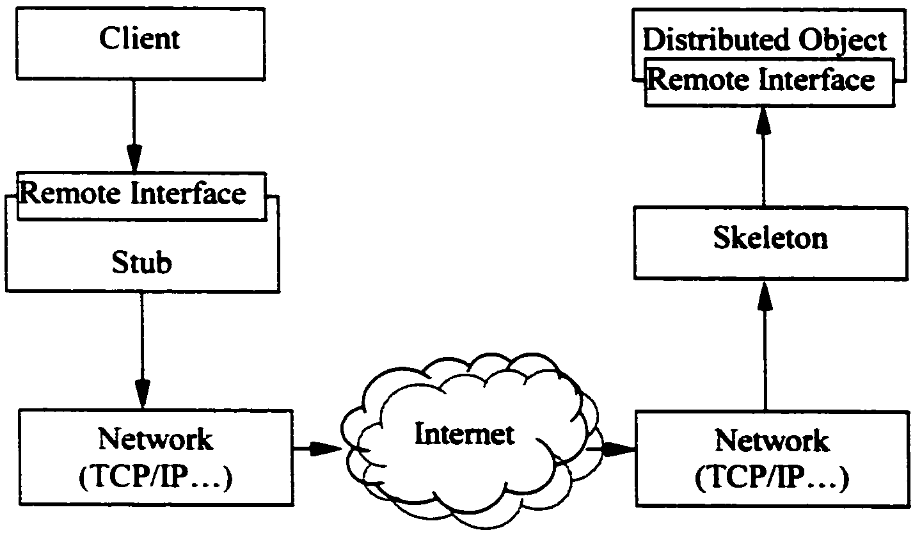


Figure 2.1 Distributed Object Model

Client/server systems cannot easily take advantage of new technologies, such as network computers. A classic client/server application is a fat client, in which the client has to have run-time environment. The client program accesses an RDBMS, shared files on a file server. or both. The client/server system can off-load some of the business logic from clients onto a series of unattended computers; the extra processing tier gives the client/server application some of the attributes of a Web-based application. In the case of poorly designed web-based application, a substantial amount of business logic is performed in Java-Script or Java applet [Manol99].

In large enterprise environments, the performance of a two-tier architecture client/server usually deteriorates as the number of online users increases. This is primarily due to the connection process of the DBMS server. The DBMS maintains a thread for each client connected to the server. Even when no work is being done, the client and server exchange keep-alive messages on a continuous basis. If something happens to the connection, the client must reinitiate the session [Singh99].

In heterogeneous network computing, influencing how any system will behave in production are its performance. reliability, network traffic, administrative, security, and workload balance characteristics. To complicate matters further. these factors are interrelated.

Many middle tier software try to solve this kind of problem. Some of them, such as ORB (Object Request Broker). make a certain kind of distributed computing available. However. there is no concept of an "operating system", where system-level functionality is handled automatically. The lack of implicit system-level infrastructure places an enormous burden on the application developer [Monso00]

Some database vendors provide transaction processing and some kinds of system level capabilities. Even this. it is not a framework and does not work well in heterogeneous network environment. Distributed object architecture is required in such an environment.

Web-based applications represent a move away from the empowering effect of desktop computing toward centralized computing. A classic web-based application uses a browser for data presentation, WASs for business logic, and database servers for storage. The WAS provides naming, security, transactions and workload management [Brown01].

A heterogeneous computing environment, in which different types of processing resource and interconnection technologies are effectively and efficiently used, has the potential to maximize performance and cost effectiveness of a wide range of scientific and distributed applications. Scalability is important for distributed computing. Scalability is one of those things that developers do not always consider when they start a project, but that becomes a major issue when it is successfully implemented [Subra00].

2.2 Web Information System Architecture

The Web and Web-based protocols (HTTP, HTML, XML, SOAP, etc.) are pervasive, they penetrate corporate firewalls, and they provide a strong infrastructure that spans consumer-to-business, business-to-business, and intra-enterprise types of relationships. The Web application topology consists of the following items:

Internet and Intranet Clients: These clients communicate with WASs through using Internet standard protocols, such as TCP/IP, HTTP, and HTML, to access business logic and data. The primary role of the client is to accept and validate user input, and to present results received from the WAS to the user.

Infrastructure Services: These provide the WAS and its business logic components with directory and security services. Included in these services are firewalls that prevent an organization's network from exposure when connecting to the Internet and prevent unauthorized access to internal data and computing resources.

Web Application Servers: These are the hubs of the web application topology processing requests from clients by orchestrating access to business logic and data on the server and returning web pages composed of static and dynamic content back to the clients. The WASs provide a wide range of programming, data access, and application integration services for developing the business logic part of a web application.

External Services: These consist of existing mission-critical application and data within the enterprise as well as external partner service, such as payment services, financial services, and external information services. Most often, these existing applications and services control the company's core business processes.

To develop a successful web application, in addition to designing the system architecture and doing software engineering, various skills are needed, such as artistic, production and business organization [Fred01]. Usually, the project is required to be available on the market in a short time. To develop a large-scale Web distributed project successfully, it's hard to imagine without using component technology.

2.3 Components and Enterprise Platform Technology

Components are objects with additional capabilities that enable them to function in large-scale information system. These additional features include the ability to do the following tasks [IBM, Maten01]:

- Create and destroy objects
- Provide an object with a location in a distributed network
- Establish a system identity for an object
- Store the state of an object in a resource manager
- Handle the activation and paging of an object
- Map transaction semantics to an object's state transitions
- Coordinate an object's locks with its underlying data store
- Control access to an object
- Search for an object
- Notify an object that an event has occurred
- Transport the state of an object across a distributed network

Server-side components can be bought and sold as independent pieces of executable software. They conform to a standard component model and can be executed without direct modification in a server that supports that component model. Server-side component models often support attribute-based programming, which allows the runtime behavior of the component to be modified when it is deployed, without having to change the programming code in the component. The server administrator can declare a server-side component's transactional, security, and persistence behavior by setting these attributes to specific values. In the following we briefly discuss the existing component models:

2.3.1 CORBA

The Common Object Request Broker Architecture (CORBA) specification defines communications between distributed objects and integrated object services [Siege00]. CORBA provides a common framework for developing applications that use components. It defines the software services that enable objects to communicate transparently across a distributed computer network. CORBA is language and platform independent and can handle objects that are implemented with different vendor packages, located on different machines, coded in different programming languages, and run on different operating systems.

CORBA specifies the common object services, such as naming, life cycle, transaction, security, persistence, which are used to manage distributed components.

2.3.2 Servlets

Java servlets are a complementary technology to applets. A servlet is a Java program that runs on a Web server. A special interface allows it to receive requests from a web browser, such as a user-initiated submission from an HTML form. Servlets typically access enterprise resources in response to an incoming request, and then format new HTML pages dynamically for transmission back to the browser. To simplify HTML generation, servlets can use Java Server Page (JSP) scripting [Subra00].

Servlets use the Java Servlet Application Program Interface (API) and its associated classes and methods. Servlets can also use Java class packages that extend and add to the Java Servlet API. Like applets, servlets are part of the Java standard and are designed to run across different platforms [SUN]. However, unlike applets, Java servlets do not use the Java Virtual Machine (JVM) supported by a browser. Servlets run on a web server. This makes their behavior more reliable and predictable.

The servlet-based application retains Internet access and the ability to process logic locally at the client. It reduces the costs of administration and client hardware. Compared to the applet-based application, the servlet-based application avoids the overhead associated with downloading an applet. It reduces concerns related to the capabilities of the client browser, since it does not require a JVM. It also avoids restrictions for communicating through a firewall.

Unlike the widely used Common Gateway Interface (CGI) programs, which require an entire process to handle user requests, servlets can handle user requests by using threads. This capability makes servlets much more efficient than CGI programs [Calla01, Subra00].

A servlet can be loaded automatically when the web server is started, or it can be loaded the first time a client requests its services. After being loaded, a servlet continues to run, waiting for additional client requests. A servlet can perform a wide range of functions including [SUN]:

- Handle connections with multiple clients, accepting input from and broadcasting results to the multiple clients. A servlet can handle multiple clients at the same time.
- Create and return an entire HTML Web page containing dynamic content based on the client request.

- Create part of an HTML Web page that can be embedded in an existing HTML page
- Communicate with other resources, including databases and Java-based applications.
- Open a new connection from the server to an applet on the browser and keep the connection open. allowing many data transfers on the single connection. The applet can also initiate a connection between the client browser and the server, allowing the client and server to easily and efficiently carry on a conversation. The communication can be through a custom protocol or through a standard such as IIOP.
- Filter data by MIME type for special processing, such as image conversion and server-side includes (SSI).
- Provide customized processing to any of the server's standard routines. For example. a servlet can modify how a user is authenticated.

2.3.3 Java Server Pages (JSP)

The function of JSP is based on Sun Microsystems JavaServer Pages Specification. JSP files are similar in some ways to server-side includes that compose of n static HTML files. because both embed servlet functionality into the web page [Wahli00]. However. in a server-side include. a call to a servlet is embedded within a special servlet tag; in JSP pages. Java servlet code (or other Java code) is embedded directly into HTML page.

One of the many advantages of JSP pages is that they enable one to effectively separate the HTML coding from the business logic in the web pages. JSP pages can be used to access reusable components, such as servlet, Java beans, enterprise beans, and Java-based Web applications.

2.3.4 Enterprise Java Beans

Enterprise Java Beans (EJBs) [Maten01, Monso00, Roman99] are the standard component architecture for building distributed applications in the Java programming language. They are server-side components that must reside in a home environment and run in an execution environment. The execution environment must provide run-time service such as transaction support, persistence, and resource management.

Handling multiple users, transactions, and allowing the application to scale are parts of a business framework that are much more generic than the business processes themselves. The enterprise JavaBeans specification tries to free the application developer from dealing with low-level issues such as multi-threading, caching, concurrency control, and resource management. More detail will be covered in the next chapter.

2.3.5 Jini and JavaSpaces

Jini network technology provides simple mechanisms, which enable devices to plug together to form a community without any planning, installation, or human intervention. Each device provides services that other devices in the community may use [SUN11]. These devices provide their own interfaces, which ensures reliability and compatibility.

Jini technology uses a lookup service with which devices and services register [Jini]. When a device plugs in, it goes through an add-in protocol, called discovery and join-in. The device first locates the lookup service (discovery) and then uploads an object that implements all of the services' interfaces (join).

To use a service, a person or a program locates it using the lookup service. The service's object is copied from the lookup service to the requesting device where it will be used. The lookup service acts as an intermediary to connect a client looking for a service with that service. Once the connection is made, the lookup service is not involved in any of the resulting interactions between that client and that service.

It does not matter where a service is implemented. Compatibility is ensured because each service provides everything needed to interact with it. Thus, there is no need for a central repository of drivers, or anything similar.

JavaSpaces employs a mechanism for accessing and processing distributed objects [SUN]. JavaSpaces manages features such as object processing, sharing, and migration. A client application makes contact with a JavaSpaces server. The client asks for a certain type of object by sending a template describing what the object looks like. The space will then respond with the entry that best fits the template description. The client sends a read operation to make a copy of the entry within that space and work with it. Alternatively, the client can also do a take operation to take an existing entry from that space, removing it from the list of entries. The client then processes the entry object as needed. Once completed, it can do a write operation to put that entry object back into the space, so that others can use it. When multiple clients looking for a particular entry, the server system needs a queue for processing.

Services employ a proxy, which is an object with service attributes and communication instructions, to move around the network. Through the processes of discovery and join, services are found and registered on a network.

2.4 Summary of Component Technology

We can see from above that all distributed object services use a naming service of some kind. Java RMI and CORBA use their own naming services. All naming services do essentially the same thing: object binding and a lookup API. Object binding is the association of a distributed object with a natural language name or identifier. A binding is really a pointer or an index to a specific distributed object, which is necessary in an environment that manages hundreds of different distributed objects. A lookup APIs provides the client with an interface to the naming system. We do some comparisons of these components in the followings.

2.4.1 EJB and CORBA

As a platform-independent and language-independent distributed object protocol, CORBA is often thought of as the superior protocol. However, CORBA suffers from some limitations. Pass –by-value, a feature easily supported by Java RMI-IIOP, was only recently introduced in the CORBA 2.3 specification and is not well supported

[Monso00]. Another limitation of CORBA is casting remote proxies. In Java RMI, user can cast or widen a proxy's remote interface to a subtype or supertype of the interface, just like any other object. This is a powerful feature that allows remote objects to be polymorphic.

EJB would not be complete without a way to integrate with CORBA however; EJB inherits many useful properties form CORBA. CORBA allows EJB customers to communicate with existing CORBA applications, as well as integrate with existing investments written in non-Java languages such as C++ and COBOL. Indeed, CORBA and EJB are related, much of the conceptual foundation in Java 2 Platform, Enterprise Edition came from CORBA [Monso00].

2.4.2 EJB and Jini

EJBs can be deployed within any application server that supports the EJB Container APIs. These containers shield the EJB developer from much of the complexity normally faced by the writers of common business logic. This increases overall developer productivity, and enhances the robustness of the resulting code as we mentioned before.

Jini technology is focused on a different problem space. It provides for the unification of a collection of services into a "Jini Community" which gives them a common device namespace, failure mode and security model [These01]. Such Jini Community is both customizable for a given user and extremely fluid, as Jini devices can both add themselves when powered up, alert a network manager, and remove themselves when hardware problems occur. There is no operator intervention of any kind required.

Java client platform allows developers to write software that is independent of both the hardware and operating system it runs on. EJBs extend this paradigm to the server by providing additional independence from various legacy infrastructures such as messaging middleware, transaction support, naming & directory services, object protocols and relational databases. Since these vary infrastructures are used to construct a corporate enterprise, with EJBs, write it once, it runs everywhere, and it integrates with everything.

Jini on the other hand redefines the client by allowing the devices it accesses to be both remote and dynamic. Essentially Jini extends the Java paradigm to make the client's device configuration independent of network topology. The network in effect becomes the client computer [Jini].

Jini offers services. as collections of objects written in Java, which can be federated to accomplish a task. Services communicate through protocols, such as discovery and lookup. look up for invocation or location of a service and discovery for registration of a service in the Jini lookup service. However. Jini is not a distributed system. and the concepts and facilities Jini uses are very limited [Carva00]. Jini has the notion of services and the facilities for finding them, allowing very simple devices to belong to the Jini communities. Jini has no notion of global resource management, no model for large scalable systems.

Further. there are not as many commercial Jini products as J2EE supports. Currently. Jini as a framework is not complete enough to enable efficient development of enterprise software. Table 2.1 provides a comparison of Jini and EJB [These01].

Comparator	**Jini**	**EJB**
Architectural Model:	Peer-to-Peer Variable-size Client Variable-size Services	3-tier Thin Client Thick Services
Typical Platforms:	Java Client and Java Device	Java Application Server and Java Legacy Server
Typical Topology:	Distributed Client	Distributed Enterprise
Wire Protocols:	RMI, Device Discovery/Join	RMI-IIOP (CORBA)
Best Feature:	Intelligent Devices use their Intelligence	Write an EJB once, Run it in every Enterprise

Table 2.1 Comparisons of Jini and EJB

The component technologies discussed above are fundamental parts for constructuring enterprise application framework.

2.5 The Extensible Markup Language (XML)

XML is used more and more popular. XML is designed to make it easy to exchange documents and structured information over the Internet. It is a standard for defining document markup languages [Brad00]. A markup language is a set of elements or tags that have one or more of the following functions [Cepon99]:

- Describing the structure of the document
- Describing the content of the document
- Controlling how the document is presented to the user.

HTML is the most widely used markup language for Web-based documents, but it has many limitations. HTML tags describe the visual presentation of Web pages and do not really specify their logical structure. HTML users are restricted to a relatively small set of tags and cannot create their own tags because commercially available Web browsers do not support tags that are not part of the HTML standard.

XML overcomes these limitations. XML users can define their own custom tag set, or use tags from any publicly available document type definition (DTD). XML tags specify the content and structure of a document. Presentation is decoupled from the document's content.

Chapter 3 Enterprise Development Platform

Enterprise platform is an enterprise application development framework or environment, such as J2EE platform, Microsoft .Net enterprise platform. The framework speeds the software development cycle. Since the application developers do not need to reinvent the system-level functionality [Monso00].

3.1 Concept of Framework

A framework is an abstracted collection of classes, interfaces, and patterns dedicated to solving a class of problems through a flexible and extensible architecture [Govon99].

Frameworks have been defined in many ways:

"A framework is a collection of classes that provide a set of services for a particular domain; a framework thus exports a number of individual classes and mechanisms which clients can use or adapt." [Booch]

"A framework is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions." [Talig]

"A framework is more than a class hierarchy. It is a class hierarchy plus a model of interaction among the objects instantiated from the framework." [Lewis]

These definitions indicate that a framework can include more than simply a collection of classes, and may also present a defined process or method of interaction between its objects. Combining objects and interactions provides a truly useful environment in which categories of problems can be addressed in far less time than it would take to reanalyze the problem and reinvent new solutions.

Frameworks encapsulate critical design architectures specific to their purpose. In doing this, classes of frameworks can be utilized by developers to save time otherwise wasted by reinventing common application problems.

3.2 J2EE Platform

The J2EE specification provides a complete range of enterprise-class functionality for server-side computing [Sun10]. The platform is designed to provide an integrated Java application environment for building enterprise-level n-tier Java applications.

The J2EE standard embraces existing resources required by multitier applications with a unified. component-based application model. This enables the next generation of components. tools. systems, and applications for solving the strategic requirements of the enterprise [SUN].

The J2EE specification requires application servers to support a specific set of protocols and Java enterprise extensions. This ensures a consistent platform for deploying J2EE applications. The EJB specification enables developers to create server-side, distributed objects in a consistent way. The specification also defines a deployment strategy so that EJBs can be deployed without regarding to any specific platform or EJB server [Nilss00]. J2EE application servers provide the following "standard" services [Monso00]:

Enterprise JavaBeans 1.1

J2EE products must support the complete specification

Servlets 2.2

J2EE products must support the complete specification

Java Server Pages 1.1

J2EE products must support the complete specification

HTTP and HTTPS

Web components in a J2EE server service both HTTP and HTTPS requests. The Servlets specification itself only requires support for HTTP. The J2EE product must be capable of advertising HTTP 1.0 and HTTPS (HTTP 1.0 over SSL3.0) on ports 80 and 443 respectively.

Java RMI-IIOP 1.0

As was the case with EJB 1.1, only the semantics of Java RMI-IIOP are required; the underlying protocol need not be IIOP. Therefore, components must use return and parameter types that are compatible with IIOP, and must use the PortableRemoteObject.narrow() method.

JavaIDL

Web components and enterprise beans must be able to access CORBA services hosted outside the J2EE environment using JavaIDL (Interface Definition Language), a standard part of the Java 2 platform.

JDBC 2.0

J2EE requires support for the JDBC 2.0 Standard Extension, but not the JDBC 2.0 Optional package.

JNDI 1.2

Web and enterprise bean components must have access to the Java Naming and Directory Interface (JNDI) Environment Naming Context (ENC), which make available EJBHome objects, JTA UserTransaction objects, JDBC DataSource objects, and optionally Java Messaging Service (JMS) connection factory objects

JavaMail 1.1 and JAF 1.0

A J2EE products must provide access to the JavaMail API for sending basic Internet mail messages (the protocol is not specified) from web and enterprise bean components. JAF is the Java Activation Framework, which is needed to support different MIME types and is required for support of JavaMail functionality.

Java Transaction API 1.0

Web and enterprise bean components must have access to JTAusertransaction objects via the JNDI ENC under the "java:comp/UserTransaction" context. The UserTransaction interface is used for explicit transaction control.

Java Messaging Service 1.0

J2EE products must support the JMS API definitions (base classes and interfaces), but are not required to provide a JMS implementation. This means that JMS is an optional service in J2EE. If a JMS implementation is supported, the connection factories can be made available through the JNDI ENC.

Currently, EJB 2.0, Servlet 2.3 and JSP 1.2 become new standard. The specification is continuing to upgrade. With simplicity, portability, scalability and legacy integration, J2EE is the platform for enterprise solutions.

3.3 Using Enterprise JavaBean

The power of EJB is that it can take advantage of the services that the container provides. Although the application developer can provide some of the same services as an Enterprise JavaBeans container, each of those services must be developed and integrated separately. If changing business environment imposes new requirements on an application, those new requirements must be met with custom code. Since Enterprise JavaBeans are written to an industry-standard API, they can often be run unmodified in a new application server that provides increased functionality [Subra00].

3.3.1 Introducing EJB

Enterprise JavaBeans (EJB) is an architecture specified by Sun Microsystems and published for the first time in March 1998. Sun Microsystems' definition of Enterprise JavaBeans is:

"The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification".

The EJB specification, which is based on the Java platform and part of Java 2 Enterprise Edition (J2EE), is a part of the framework for server-side component-based development. EJB is used for building server-side systems that are scalable, reliable, secure, and portable. The EJB specification also defines a set of services that must be provided by EJB servers, such as transaction management, persistence, security, concurrency and instance pooling.

Transaction is an important concept in distributed computing. A transaction is a set of operations that transforms data from one consistent state to another. This set of operations is an indivisible unit of work, and in some contexts, a transaction is referred to as a logical unit of work. A transaction is also a tool for distributed systems programming that simplifies failure scenarios. Transactions provide the ACID properties [IBM]:

- Atomicity: a transaction's changes are atomic: either all operations that are part of the transaction happen, or none happen.
- Consistency: a transaction moves data between consistent states
- Isolation: even though transactions can run concurrently, no transaction sees another's work in progress. The transactions appear to run seriously.
- Durability: after a transaction completes successfully, its changes survive subsequent failures.

The EJB specification provides a detailed description of the services needed to support enterprise beans. It separates the enterprise bean's business logic from the intricacies of persistency, transactions, security, and other middleware-related services.

The architecture of EJB is shown as Figure 3.1. An Enterprise Java Server (EJS) manages one or more containers.

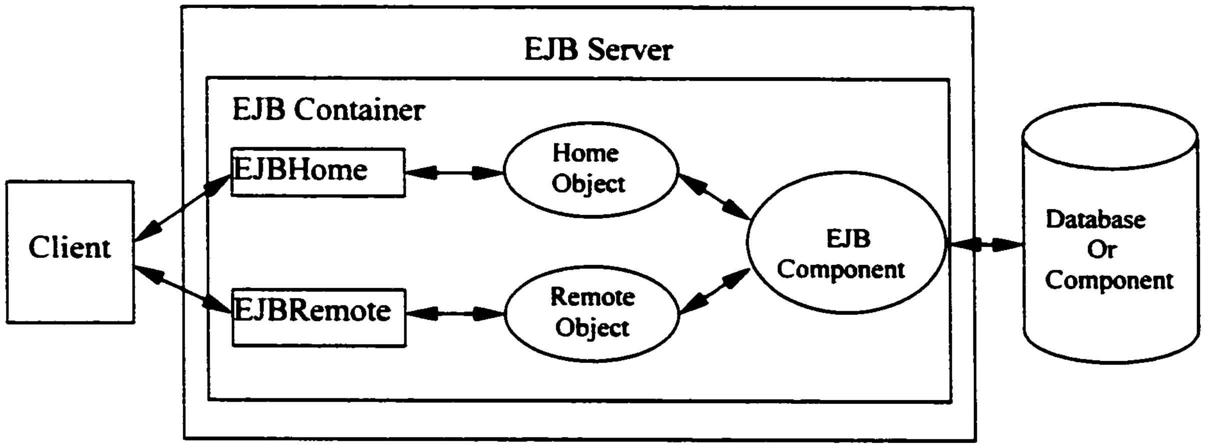


Figure 3.1 EJB Architecture

EJB containers provide services to a set of enterprise bean instances. First, a container controls the enterprise bean's life cycle: it creates beans instances, manages pools of instances, and destroys them. Second, containers are also responsible for providing services such as persistency and security to the beans they manage. Third, containers are transparent to the programs. EJB client can always use the services of a container when invoking an enterprise bean. The container take care of providing the level of service the client requested at deployment time.

A container is the run-time component of an EJB environment. It communicates with the enterprise server to implement naming (JNDI), transaction, security and memory management services, and obtains access to the service on behalf of the enterprise-bean instance. The EJS can be a WAS or a database server.

EJBs are distributed objects [Maten01, Roman99]. A client access EJB by EJBHome and EJBRemote interface. EJBHome lists the methods that client programs can use to create and find bean instances. The EJB Remote interface defines the business methods of the enterprise bean. The container creates the EJB Object class from this interface definition. After a client program has access to the home object, it asks the container to create an enterprise-bean instance. A client program never accesses a bean instance directory; it always goes through the EJB Object. A bean that has been idle for some time can be transparently swapped to disk and re-activated when the client program requests it. This would not be possible if the client held a reference to the bean instance. The steps involved in using an EJB is shown in Figure 3.2 [IBM]:

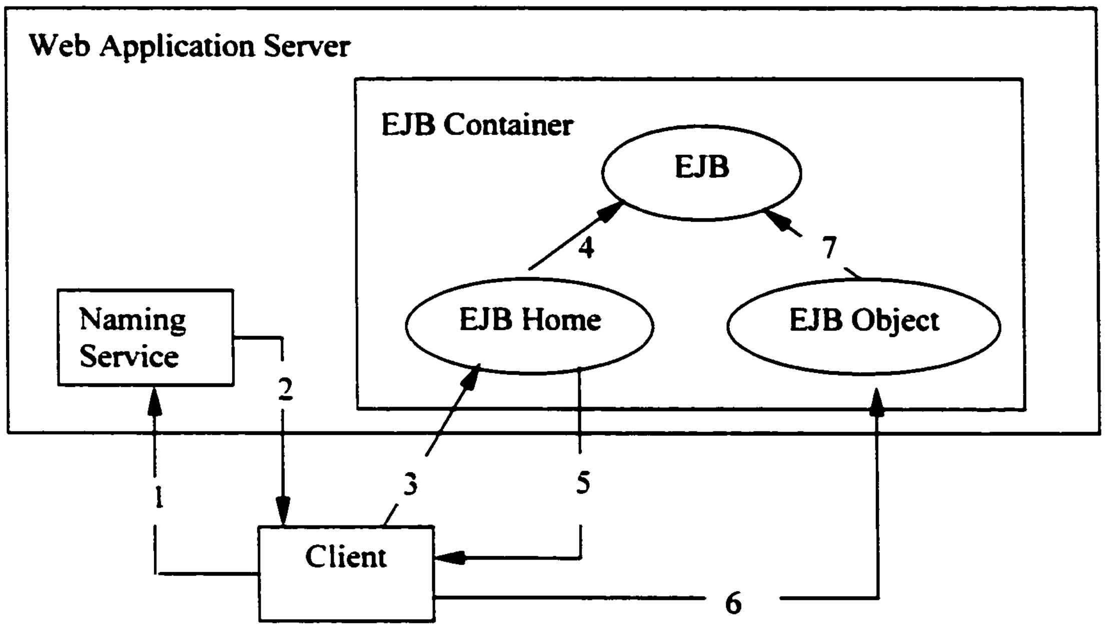


Figure 3.2 Steps for Using an EJB

Steps used to call methods on an EJB [IBM]:

1. The client requests from the naming services a reference to the EJB Home interface of a particular class of EJBs.
2. The naming service replies with the location of the Home interface instance for the EJB class in the container in which the EJB is deployed.
3. The client performs either a create (for the new bean instance) or a find (for an existing entity bean instance) on the EJB Home interface instance
4. The EJB Home interface instance locates or creates the EJB instance and places it in the container and creates the EJB Object interface instance.
5. The EJB Home interface instance replies to the client with a reference to the EJB Object instance.

6. The client calls methods on the EJB Object interface instance to access business logic on the EJB.
7. The EJB Object interface instance calls the corresponding methods on the EJB while the container manages the resource needed to accomplish this task.

There are three types of EJB: Session bean, Entity bean and Message-Driven bean. Session bean can be divided as stateless session bean and stateful session bean. Session bean represent tasks and operations. They are transient and do not correspond to data in persistent storage. A stateful session bean can maintain state information. State information that is associated with a stateful session bean is not shared between clients. Entity beans represent persistent data. Each entity bean is mapped to a data store that is shared with other enterprise bean. In most cases, an entity bean must be accessed in some transactional manner. Entity beans are unique and do not store state information. They can be shared between clients.

Management of the persistent state of an entity bean can be done in two ways [IBM]:

- Bean-managed persistence (BMP)
- Container-managed persistence (CMP).

In BMP, EJB directly accesses persistent storage. Accessing to persistent storage must be implemented by the software developer. In CMP, the bean relies on its container to provide transparent access to persistent storage. The software developer does not need to explicitly implement access to persistent storage.

The properties of EJB can be defined in the EJB description file. Access control or security, bean pool size and bean type can all be defined. EJBs have states. EJB state transition diagram is shown in Figure 3.3 [IBM].

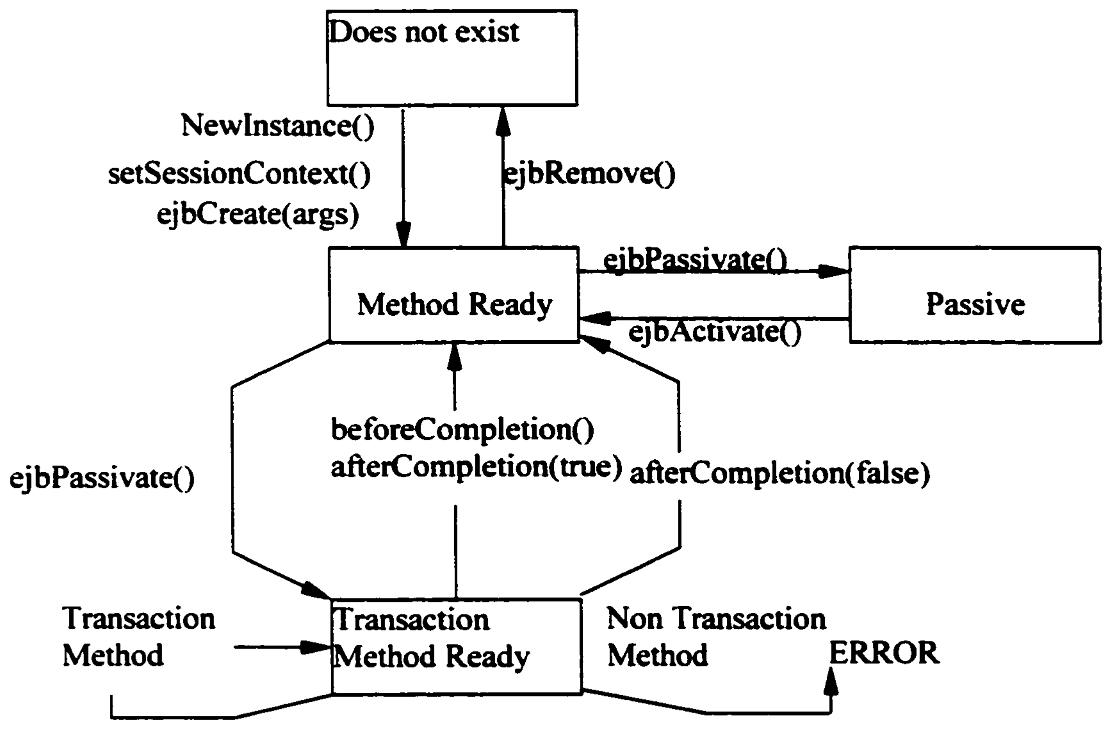


Figure 3.3 EJB State Transition Diagram (Stateful EJB)

The entity beans and stateless beans can be pooled [Gomez00]. The benefit is that the pool of beans can be much small than the actual number of clients connecting. This is due to client "think time", such as network lag or human decision time on the client side. While the client is thinking, the container can use the bean instances to service other clients, saving previous system resources.

Another benefit of EJB is that developers do not need to write thread-safe code [Monso00]. The developers never need to worry about thread synchronization when concurrent clients access the component. The EJB container will automatically instantiate multiple instances of the component to service concurrent client requests.

The container thread services can be both a benefit and a restriction. The benefit is that application developers do not need to worry about race conditions or deadlock in the application code. The restriction is that some problems lend themselves very well to multithreaded programming, and that class of problem cannot be easily solved in an EJB environment. EJB is intended to relive component developers from worrying about threads or thread synchronization. The EJB container handle those issues for developer by load-balancing client requests to multiple instances of a single-threaded component. An EJB server provides a highly scalable environment for single-threaded components.

EJB masks the physical location of the remote object the client is invoking on. Location transparency is a necessary feature of multi-tier deployments. It means that client code is portable and not tied to a specific multi-tier deployment configuration. Also, EJB specification use RMI-IIOP that is more robust than RMI in distributed communications.

EJB's security management [Vogel99] is very powerful as it gives the developer the ability to create the components without having to hard code security issues. The developer uses an abstraction called security role. A security role is a grouping of user permissions that enable the execution of the various components that make up an application. The security roles are assigned at deployment time according to the corresponding security policies of the operational environment. The JavaBeans component architecture demonstrated the usefulness of encapsulating complete sets of behavior into easily configurable, readily reusable components on the client side. The convergence of these three concepts -- server-side behaviors written in the Java programming language, connectors to enable access to existing enterprise systems, and modular, easy to deploy components -- led to the EJB standard and its ability to effectively leverage and consolidate industry-wide knowledge of middleware. EJB technology helps protect existing IT investments and promotes freedom of choice for future investments [SUN].

3.3.2 How to use EJBs

The clients of EJB can be a servlet, JSP, or another EJB. The clients communicate with session beans, which can be stateful or stateless. These beans then communicate with entity beans, which in turn communicate with the data store.
The typical use of EJB can be shown as Figure 3.4

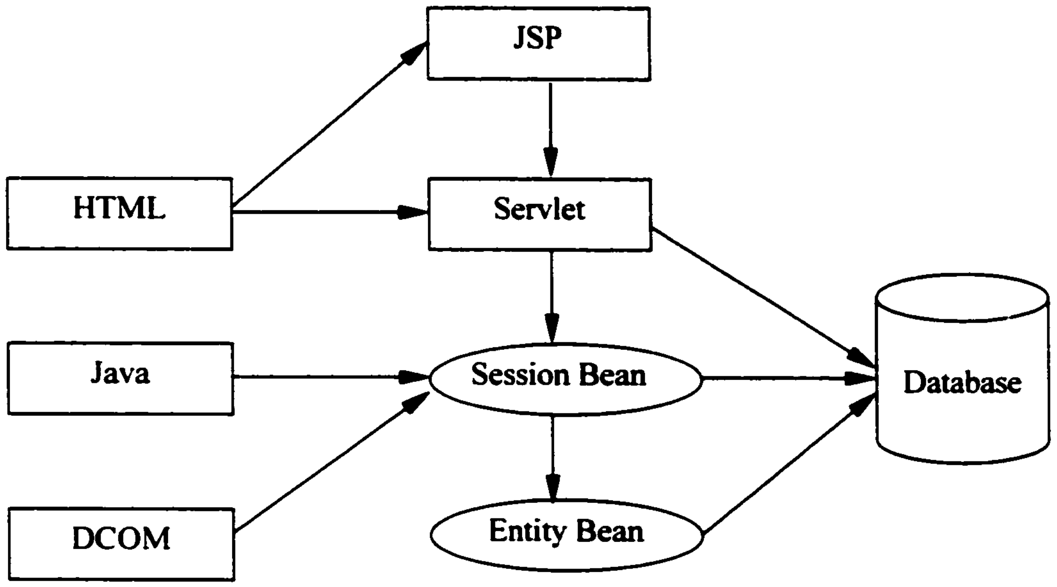


Figure 3.4 Calling EJB

3.4 Review of Previous Research

In this section, we will introduce and analyze CodeNet and several computational grid application systems.

3.4.1 CodeNet

The intention of building CodeNet [Zhong00, Zhang00] is building a class library for software reuse. The system includes code generator, language analyzer and Oracle8 Database. Using CORBA as the middleware, client and server can communicate through Internet. Basically, it's single client/server model, since it has single server and multiple clients cannot connect to the server concurrently.

The system provides a simple function for natural language processing and code generating. It gives a demonstration for reuse-oriented program developments. Besides some bugs relative to operating system, the system is not complete. There are some serious limitations such as single database connection. This makes the system not only limited to single user connection, but also the server to database and user to database operations cannot be concurrently processed. Even for single server, the powers of the server and database server are far less utilized. More detail can be referenced from [Zhong00, Zhang00].

3.4.2 Some Examples of Current Grid Computation and Multi-Server Technology

Although the history of grid computation is not long, there are some application systems and projects.

Legion [Legion] is developed by University of Virginia. It connects workstations, supercomputers, and other computer resources together into a system for Meta computing. One application project is Seti (Set it at home) project. Many home computers are not using during the night. The system sends the data from Harbor telescope to those home computers through Internet for scientific calculation. The data need to be segmented to small blocks before they are sent to the individual home computers. The home computers process it and send the results back to the system. This benefits both home computer users and scientific research. However, the data blocks between the individual home computers have not much relations.

Globus [Globus] is an object-oriented distributed system built on a stationary communication infrastructure whose emphasis is on high availability of reliable objects. Processes interact and communicate through shared objects whose state can physically distributed. Each communicating process maintains a copy of the shared object manages the state among the replicas of the object. However, "Globe is not a user-oriented OS" [Carva00]. Its emphasis is on objects and on their policies for distribution, replication and coherence. Globe does not provide the view of mobile, customizable and user-oriented execution environments.

Globus and Legion are pursuing the goal of building software architecture for grid environments. "These systems do not address much for user, component, mobility and change-oriented philosophy "[Carva00].

The FETISH system is trying to communicate with Anyone, Anywhere, Anytime [SUN]. It uses Jini technology to allow businesses to register its service, data or application with FETISH and receive access to any other service on the network.

The FETISH project is enabling a services-to-services (S2S) capability that will allow end users to integrate multiple services together, code them to interact with one another and develop complex value-added packages and services to others.

The architecture is designed as a peer-to-peer (P2P) distributed object system, allowing the FETISH servers to establish more options for services and data to navigate across the network. The FETISH servers act as proxy, lookup devices and have a directory of all available services and their APIs. The actual services, data and applications do not actually reside on the FETISH servers; requests are mapped to the closest available services.

We have explained the framework concepts and introduce EJB technology in this chapter. In the next chapter, we will discuss several possible approaches of using EJB in multi-server model design.

Chapter 4 Using EJB in Multi-Server Environment

When considering in multi-server environment, things become much more complicated than in a single server environment. We need to solve resource sharing, concurrency, multiple protocol, security, parallel computing, persistence, isolation, fault tolerance, and load balance that are much simpler or do not even exist in single server environment. Multiple servers work together to compose a cluster [Buyya99]. There are two purposes:

1. Clustering for Performance

These types of clusters are typical for high performance computing (HPC). Workloads focus on performance and scalability by applying as many CPUs to a problem solving as possible. Most scientific calculations use some form of batch-processing or work-sharing software. There are usually no resilience features. In case of application or system failure, a checkpoint/resume mechanism might exist to provide the restarting of failed batch loads. Scientific clusters are used for a wide range of disciplines, such as weather forecast, genome mapping, protein folding, automobile design, high-energy physics, and vary kinds of simulation and modeling.

2. Clustering for High Availability

High availability (HA) clusters add availability features on top of operating system infrastructures. Availability is achieved by using scripts that monitor application service health on individual cluster nodes. In case of service failure (due to the failing of disks, networks, or the application service itself), the application will be restarted on another node. Individual cluster nodes are primarily administered independently of each other.

EJB is a distributed object that is callable from a remote system. An EJB has many good properties as we mentioned in the previous chapters, such as transaction, security, persistence, portable, etc. It fits very well in the distributed computing environment. Of course, EJB is not a panacea. It has its own limitation and can not solve all problems. In this chapter, we will discuss how EJB can solve HPC and HA problems. First, let us see some basic concepts.

4.1 Basic Concepts

4.1.1 Stateful and Stateless Request

In Web client-server environment, the client sent requests to the server and the server sent replies back to the client. If each client's request is completely independent of every other client's request, then it does not matter if two requests are ever processed on the same server or not, we can call them stateless request. If the request from an individual client is depend on the previous request, we can call it stateful request.

4.1.2 Workload Balance and Fail Over

Failover and load balance [IBM, BEA] are very important for improving system availability and efficiency.

Load balance optimizes the distribution of work. Incoming work requests are distributed to the application servers and other objects that can more effectively process the requests. Load balance is most effective when used in systems that contain servers on multiple machines. It also can be used in systems that contain multiple servers on a single, high-capacity machine. In either case, it enables the system to make the most effective use of the available computing resources. Further, to increase the prcessibility, it should be able to add additional machine to the server group. Adding additional machines requires logic to load-balance client requests across machines. There must be logic to load-balance incoming requests to the presentation tier, as well as load-balance requests from the presentation tier to the business tier.

Failover is sometime called fault-tolerance, in that it allows the system to survive a variety of failures or faults. Failover is only on a technique in the much broader field of fault-tolerance, and there is no such a technique that can make a system100% safe against any possible failure. The goal is to greatly minimize the probability of system failure, but it can not be completely eliminated.

Failover is the process by which the cluster automatically relocates an application from a failed node to a healthy one. When a failover occurs, clients may see a brief interruption of service, but they are not aware that the application has been rehosted on a different physical cluster node [IBM].

4.1.3 Idempotent

An idempotent method is one that can be called repeatedly with the *same* arguments and achieve the *same* results [Roman99]. It can be defined as:

if $f(x) = y$,

then $f(f(x)) = y$

An idempotent method in a distributed system is one that does not impact the state of the system, so it can be repeatedly called. It is an important concept for fail over.

4.1.4 Smart Stubs

Smart stubs [SUN] are direct replacements for the original stubs that perform the same function with additional behavior. They can be added and removed with no code changes within the client application or the server object.

The additional behavior may be the timing of method calls, caching of data, or a collection of references to multiple instances of different servers so that it supports workload balance and fail over. Smart stubs are invisible to object servers–proxies and are purely client-side facilities [SUN].

When for workload balancing and fail over, the smart stubs provide all the necessary information for failover and load balancing across multiple server environments. A smart stub has a policy that determines the specific algorithms that it must use for load balancing and failover.

Whenever the client invokes an operation on the bean, the smart stub automatically and transparently forwards the request to one of the available server objects [SUN]. When the invocation of a method fails, the policy determines if a retry should be attempted. Retries are not always necessary since it may be caused by network congestion, server slow down; but when it is possible, the policy will choose a new server offering that specific service to handle the request. Therefore it can achieve fail over.

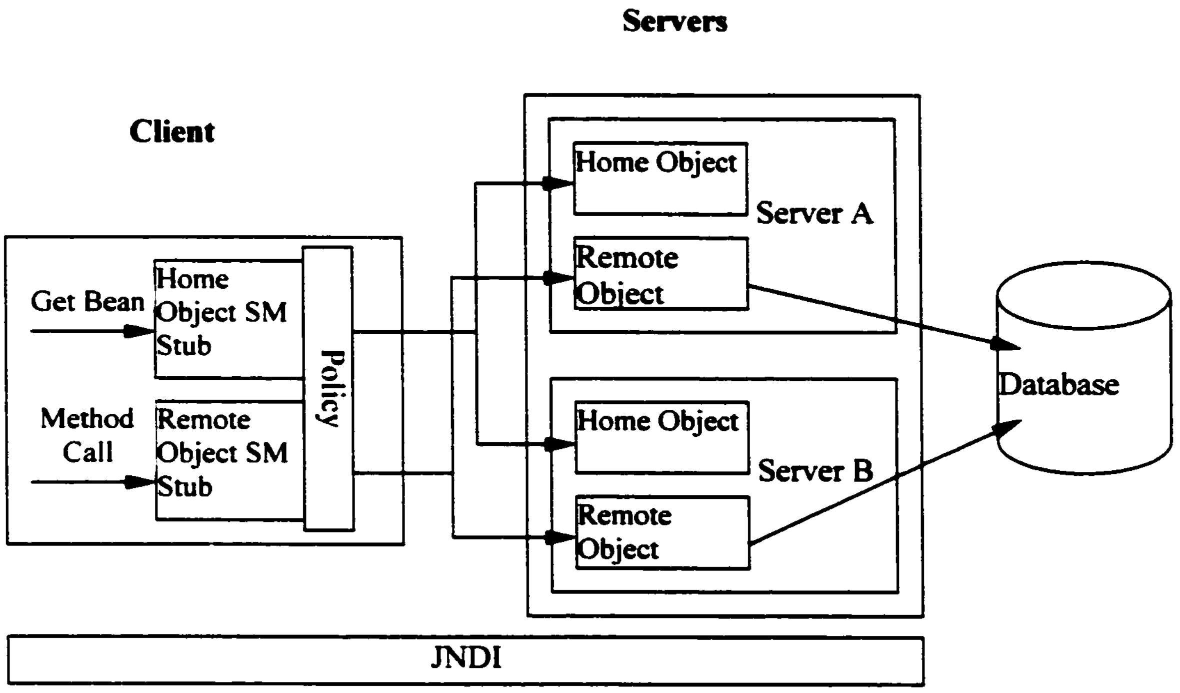


Figure4.1 Smart Stub and Policy

When a bean is deployed in a cluster, the home interface is bound into the replicated naming service. When a client looks up the home interface with JNDI, it gets a smart stub with reference to the actual home on each server that deployed the bean. When the create() or find() methods are called, the smart stub routes the call to the appropriate server based on the policy. That server will then create an instance of the bean. Figure 4.1 illustrates this.

The policy for load balancing algorithm could be round robin, random, weighted round robin, or statistically based on network modeling.

When the naming tree is replicated across all the members of a cluster, a client can request the initial context from a JNDI tree from any member of the cluster. When a server of the cluster wants to offer a service, it places a stub on the corresponding node of its naming tree. The stub is actually placed in a service pool at the node. This service pool contains all the stubs belonging to the members of the cluster offering the service. The lookup in a JNDI tree is aware of all the service providers in the cluster.

The creating of smart stub can have several different approaches. For example, idl2java compiler generates the default proxy class and three methods that are added to the interface Helper class. The developer creates the smart stub subclass by extending the default client proxy class, and overriding methods as appropriate. The default proxy handles any remote object methods that are not overridden.

Since there are two main types of EJB, session EJBs for process and entity EJBs for data, multi-server data process is actually these two kinds of EJBs process when we consider with using EJB design. We will discuss how EJBs can handle workload balance and fail over in the followings.

4.2 Stateless Session Bean

For stateless session bean, every client request directed to a stateless session is independent of any previous request that was directed to the same bean. The container will maintain a pool of instances for each type of bean, and will provide the appropriate type of the bean when it receives client's request. Because of this, it does not matter if the same actual bean instance is used for consecutive requests, or whether two consecutive requests are serviced by bean instance in the same application server.

4.2.1 Workload Balancing

Since all instances of the same type of a stateless session bean are considered identical even though they use different Java objects, all method invocations on the remote home and remote stub in different server are the same. Thus, clients' requests can be load balanced in servers. Also, developers can decide whether they want the methods on the remote stub to be load balanced across instances in the cluster or pinned to instances on a certain server. Further, the subsets of methods on a single stub can load balanced. So stateless session EJBs are the most scalable types of EJBs.

4.2.2 Fail Over

Since stateless session bean is idempotent, fail over on remote home stubs can always occur with no problem. Fail over on remote stubs can automatically occur on methods in the remote interface that are idempotent. Stateless session beans can have *create()* and *home()* methods in the home interface. Both of these methods have idempotent property.

When we want to code the non-idempotent method, we can programmed it as manually fail over, or use stateful session bean instead of stateless bean. Manual programmatic fail over can be adopted for methods that are non-idempotent and for those WASs that do not support automatic fail over of methods. The following code will manually fail over any method invocation that is not automatically done.

```
InitialContext ctx = null;
ClientHomeStub home = null;
ClientRemoteStub remote = null;

try {
  ctx = context1;
  home = ctx.lookup(context1);

  // Loop until create() completes successfully
  boolean createSuccessful = false;
  while (createSuccessful == false) {

    try {

      remote = home.create();

    } catch (CreateException ce) {
      ctx =context2;
```

```
        home = ctx.lookup (context2);
        continue;

      } catch (RemoteException re) {
        // Handle system exception here.
        // If fail over should occur, call continue;

      } catch (Exception e) {
        // Home stub failure condition detected.
        // If fail over should occur, call continue;
        continue;
      }

      // If processing gets here, then no failure condition
detected.
      createSuccessful = true;

    }  // while

    boolean answerIsFound = false;
    while (answerIsFound == false) {

      try {

        remote.method(…);
      } catch (ApplicationException ae) {
        // Handle application exception here.
        // If fail over should occur, call continue.

      } catch (RemoteException re) {
        // Handle server-side exception here.
        // If fail over should occur, call continue.

      } catch (Exception e) {
        // Failure condition detected.
        // If fail over should occur, call continue.
        continue;
      }

      // If processing gets here, then no failure condition
detected.
      answerIsFound = true;

    } // while
  }
  catch (Exception e) {}
```

Figure 4.2 Manual Fail Over Program

Clients get the required objects by searching the JNDI tree with "context". The context is simply a nickname instead of hard code of actual object. The client get actual object home interface from JNDI. If the client can not find the real object from JNDI, then client can decide to try another nick name or another JNDI tree. It can be implemented in WAS or client code.

4.3 Stateful Session Bean

Each instance of a particular stateful session bean exists in only one application server, and can only be accessed by directing requests to that particular bean. Two ways for dealing with multi-server:

1) EJB's home interface is duplicated to all servers in the same server group or cluster.

2) Store the EJB's state to the database.

It can be achieved in two ways: store the state directly or by calling an entity bean. It is shown in Figure 4.3

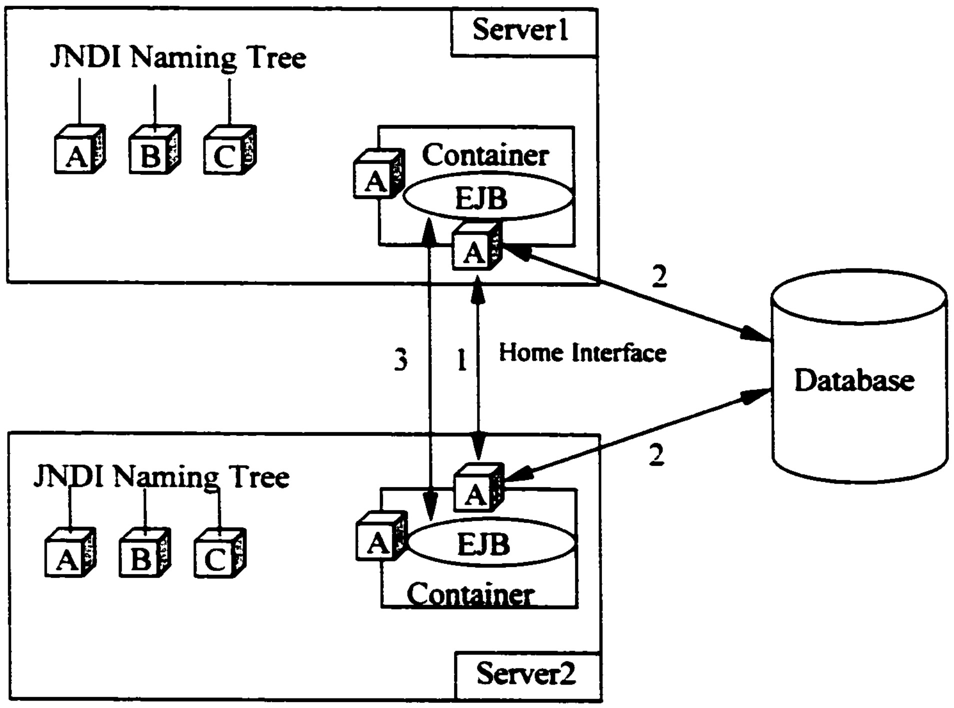


Figure 4.3 Fail Over of Stateful Session Beans

4.3.1 State Replication

For stateful session EJBs, the state could be replicated in multi-servers to resist server-side failures and loss of session-oriented state data. This state replication is very similar to HttpSession replication that is offered by many servlet engines.

The critical, transactional, and persistent data should always be stored via JDBC or entity EJBs instead of session EJBs. Stateful session EJBs should be used for storage of session-oriented (conversational) data that would not adversely impact the system if the data were lost [Roman99].

Replication of stateful data typically occurs at one of two points:

1. At the end of every method call.
2. After the commit of a transaction.

Containers that replicate the state of a stateful session EJB can have two ways: database-oriented replication and memory replication. We discuss these scenarios in the following:

4.3.1.1 Database Oriented Replication

One approach to is to keep the state in a database or other persistent store. This approach can rely on underlying database locking to avoid concurrency conflicts. This is suitable for persistent components. This approach scales like stateless services, but differs in that the latter requires explicit disk reads/writes. Some databases are very good at caching objects in memory and doing the minimal disk I/O necessary to provide transactional protection.

4.3.1.2 Memory Based Replication

A second approach is to keep a secondary copy in memory on another machine. This is more susceptible to failures and is not suitable for persistent components. The difficult here is determining when and how the state of an object has changed, since persistent components are generally just written out when transactions are committed. Stateful session beans can be configured to use in-memory replication. The replication system takes care of transporting an update data from the primary copy to the secondary copy.

Scalability comes from distributing the primaries and secondaries across the cluster. This is in contrast to replication systems that keep all of the objects on (1) a fixed-size subset of the servers, or (2) all of the servers. Approach (1) typically uses a process pair of servers to hold session state. Approach (2) requires so much "chatter" back and forth between all the server instances that after two or three nodes.

4.3.2 Workload Balancing

Since remote home stubs can invocate different servers in the cluster, these stubs can be serviced by any container in the cluster. With stateful session beans, the server is responsible for maintaining state on behalf of the client within the server. Subsequent requests from the client need to be directed to the server that is hosting the instance that has its state.

As a result of this "pinning" of clients to their server-side data objects, remote stubs are limited in the load balancing of method invocations. A remote stub must direct all requests from a client to a server that is hosting the object with the client's data. In the most common scenario, a stateful session EJB instance is only hosted on a single server, however, it is possible to replicate stateful session EJBs across servers or persistent stores. In a situation where a stateful session EJB is replicated across multiple servers, a remote stub could conceivably load balance different requests to different servers. This would not be an ideal scenario, since most servers that implement stateful session EJB replication have a designated "primary" object that requests are sent to first. The effort involved with load balancing requests in this scenario outweighs any benefits that may be achieved from caching [These01].

4.3.3 Fail Over

Stateful session beans can have *create()* and *home()* methods in the home interface. Both of these methods have stateless behavior and are idempotent. Fail over on remote home stubs can always occur automatically.

Fail over on remote stubs can automatically occur on methods in the remote interface that is idempotent. Stateful session EJBs can also have idempotent methods. Any method that does not alter the state of the system or alters the value of the state stored in the stateful session EJB is an idempotent method. For example, if a stateful session EJB has a series of repetitive get methods to retrieve the values of static data stored in the server, these get methods would be idempotent.

Automatic fail over for stateful session EJB idempotent methods can only occur under certain conditions. If a stateful session EJB instance is replicated after every method invocation, automatic fail over on idempotent methods can always occur in the server group. If replication occurs after every transaction commit, automatic failover can only occur in between transactions. Since a stateful session EJB can have its state altered during a transaction, the replicated instance would not see the updated state until the commit of the transaction. In this case, automatic fail over can only occur in between transactions. Also, we can utilize database replication method as we mentioned above to solve this problem.

Similar to stateless session beans, for methods that are non-idempotent and for those containers that do not support automatic fail over of methods, manual, programmatic fail over can be employed as described for stateless session EJBs.

```
InitialContext ctx = null;
SomeHomeStub home = null;
SomeRemoteStub remote = null;

try {
  ctx = …;

  // Automatic fail can occur by the stub.
  home = ctx.lookup(…);
  boolean answerIsFound = false;
  while (answerIsFound == false) {

    try {

      remote.method(…);

    } catch (ApplicationException ae) {
        // Handle application exception here.
        // If fail over should occur, call continue.

      } catch (RemoteException re) {
        // Handle server-side exception here.
        // If fail over should occur, call continue.

      } catch (Exception e) {
        // Failure condition detected.
        // If fail over should occur, call continue.
        continue;

      }

      // If processing gets here, then no failure condition
detected.
      answerIsFound = true;

    } // while
} catch (Exception e) {}
```

Figure 4.4 Manual Fail Over Program

This program is similar to the previous one. Client can try different JNDI, try another nick name, or whatever.

4.4 Entity Bean

Entity beans represent data. The information contained in an entity bean is not usually associated with a "session" or with handling of one client request or series of client requests. But it is common for one client to make a succession of requests targeted the same entity bean instance. It is possible for multiple independent clients to access the same entity bean instance more-or-less concurrently [Maten01].

For read-mostly data, such as static data, that does not require strict transactional semantics. We would want broadly used data to be widely replicated for the highest performance, but at the same time, cannot afford the very high cost of simultaneously updating every such instance within a single transaction. Such "read mostly" entity beans require relaxing strict transactional semantics to gain the desired efficiency.

4.4.1 Workload Balancing

Since entity beans are rarely accessed remotely, most access to entity EJBs occurs over local interfaces by collocated session EJBs and not remote interfaces by remote clients [Roman99]. The load balancing and failover aspect of entity EJBs is handled at a higher level, either by session EJBs or through web-based load balancing. The need to load balance invocations to different servers is not really an effort employed by a home or remote stub.

Some application servers do support remote home stub and remote stub clustering for entity EJBs. In terms of load balancing, entity EJB invocations follow load balancing rules similar to stateful session EJBs: remote stub invocations can be load balanced in between transaction commits.

4.4.2 Fail Over

Fail over of entity EJBs is exactly the same as stateless session EJBs. Entity beans can have create(), find(), and home() methods in the home interface. All of these methods have stateless behavior and are idempotent.

Automatic fail over for entity EJB remote stub idempotent method invocations occurs conditionally. If an entity EJB instance is persisted after every method invocation, automatic fail over on idempotent methods can always occur since all of the other servers can load the latest persistent state upon every invocation. However, if synchronization with the persistent store only occurs after a transaction commit, automatic failover can only occur in between transactions or on those entity EJBs that do not participate in a transaction.

4.4.3 Caching

Entity EJB represents data. In order to improve performance, entity EJBs are often cached [IBM]. Since the amount of synchronization needed to manage a cache in a cluster is very high, a cache generally needs to be accessed 3 or 4 times more update than without using caching to make benefits of having the cache outweigh not having it.

There are two strategies for caching:

- Single-container or Multi-container
- Read-only or Read-write [IBM]

When the data is relatively isolated, EJB can be cached in single container. In multi-container, the bean is always reloaded from the database at the beginning of each transaction [There01]. Therefore, it is in theory acceptable for a client to attempt to access the bean and start a new transaction on any container that has been configured to host that bean.

Read-only caches are easy to implement and are non-transactional while read-write caches are difficult to implement but can be transactional. Read-write caches improve some performance, but it is more expensive. Figure 4.5 shows the Read-only cache and Read-write cache.

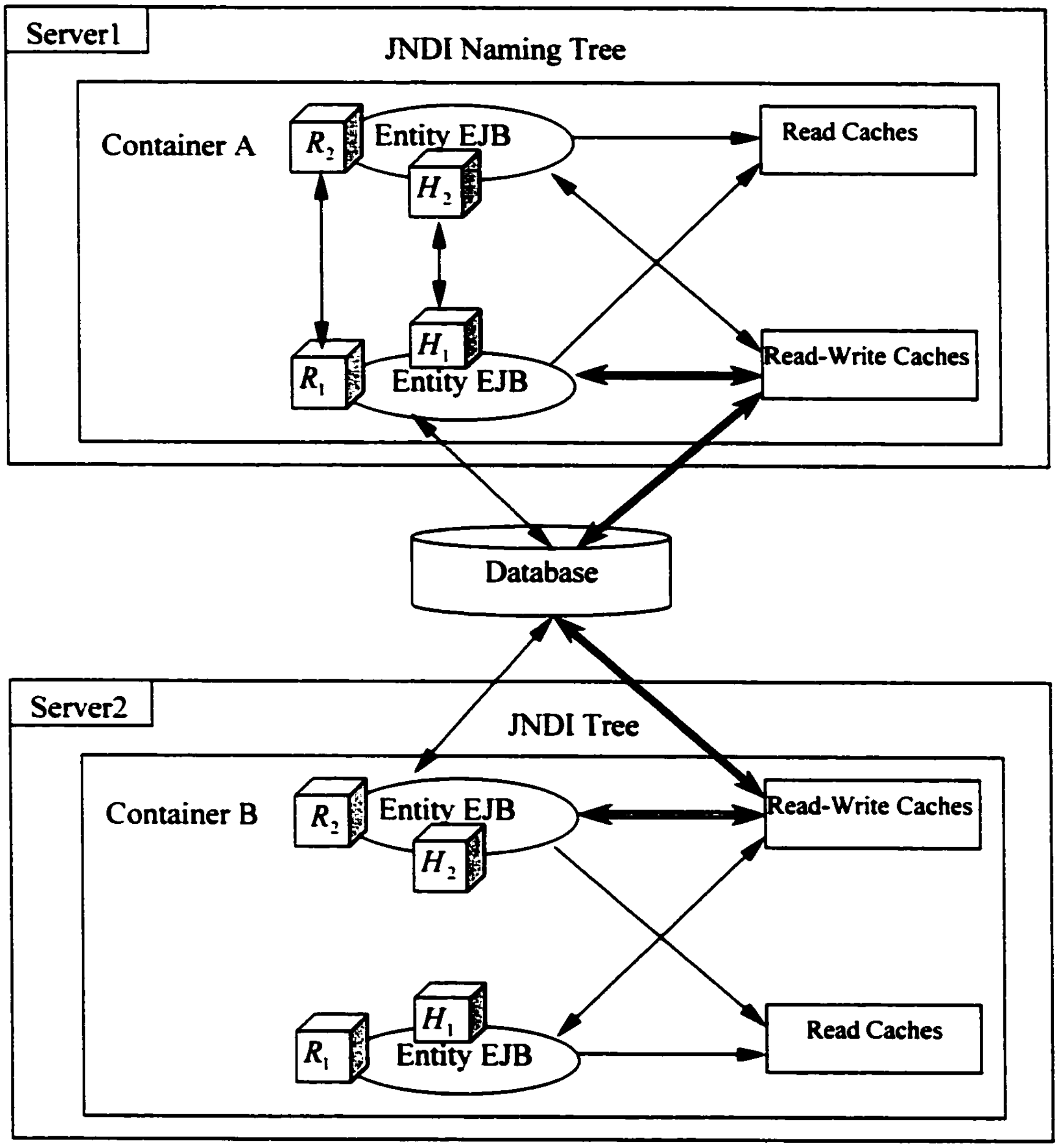


Figure 4.5 Read-only and Read-Write Caches

4.4.3.1 Read-Only Caches

A read-only cache allows for non-transactional access to the data of an entity EJB. Read-only caches implement some sort of "invalidation strategy" that determines when the data in the read-only instance is no longer valid and should be reloaded from the persistent store. Entity EJBs that operate in a persistent cache never have ejbStore( ) called and can never be called with a transactional context [Roman99].

The most common invalidation strategy approaches used on read-only entity EJBs include: timeout, programmatic through the home stub or remote stub, or system-wide notification from a read-write cache. Read-only entity EJB caches that are invalidated by a timeout cache their contents on a periodic timer basis. When the timer is triggered, the cache is invalidated and the read-only entity EJB will be reloaded immediately or upon the next method invocation. Timeout invalidation can lead to reads of inconsistent data since a transaction commit that alters the data on a read-write cache of the same primary key could have been executed before the invalidation occurs. Programmatic and system-wide notification invalidation strategies are the most common and very similar.

Read-only caches typically only use a single entity EJB instance per primary key per server. This means that multiple, concurrent accesses to the read-only cache are serialized in a single server, but concurrent across multiple servers in a cluster. Read-only caches typically do not use multiple instances for a single primary key in a single server.

Many application servers provide a "read-mostly" pattern that connects a single server read-write cache with a cluster-wide read-only cache on a programmatic invalidation strategy. When a read-write cache has a successful commit, a notification message is sent from the container to all of the read-only caches for the same primary key. Each of the read-only caches that receive the notification message immediately invalidates their cache. If the read-write cache has a transaction rollback, a notification message is not required since the state contained in the read-only cache would be identical to the rolled back state of the read-write cache. This "read-mostly" strategy provides an automatic coupling between the read-only caches and read-write caches of a single primary key [There01].

4.4.3.2 Read-Write Caches

A read-write cache across the cluster is different from a transactional read-write cache that exists on a single server. With single server read-write caching, the server does not have to coordinate the data it maintains for any in-memory primary keys with other containers caching the same primary key. Single server read-write caching relies upon database transaction isolation levels to detect collisions. A clustered read-write cache not only has to detect any collisions at the transaction isolation level, but it must also notify or propagate any changes to other caches in the cluster.

Implementing clustered read-write caches is difficult and can have different strategies. Clients that access a clustered read-write cache can do so without having to differentiate between different JNDI names. The cache determines whether or not the current access is read-only or read-write. If the data associated with the primary key being accessed is not locked by the application server or is not locked at the database level, and there is no transactional context associated with the current method invocation, the read-only cache can be accessed. Any other invocation would access the cache using read-write and transactional semantics (that may lock the data either in the server or at the database level). The two most popular read-write strategies are [These01]:

- Application server monitored database triggers. In this scenario, every time the data that maps to an entity EJB read-write cache is modified in the persistent store, a trigger is fired. When the trigger is fired, each read-write cache updates its contents so that read-only clients can access the latest data. Since each of the servers will receive the notification, the updating of the data can occur concurrently across the cluster by multicast.

- Cooperative synchronization of each read-write cache in the cluster. This strategy assumes that all data for a given primary key is read-only until locked by another server. All notifications between servers can occur using multicast, point-to-point, or a higher level of message propagation such as JMS.

4.5 EJB Clustering

In web client-server environment, a failed request could have occurred at one of three points:

1. After the request has been initiated, but before the method invocation on the server has begun to execute

2. After the method invocation on the server has begun to execute, but before the method has completed.

3. After the method invocation on the server has completed, but before the response has been successfully transmitted to the remote client.

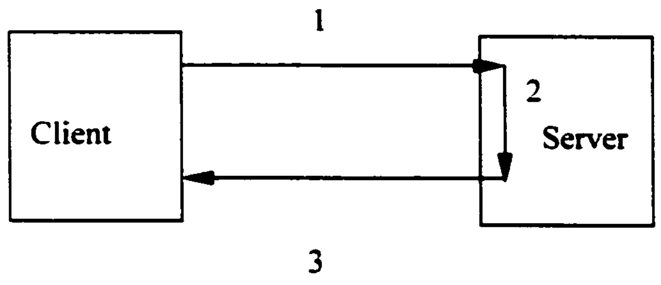


Figure 4.6 Fail Points

In the first case, fail over of the request to another server can always occur. In the second and third cases, fail over of the request to another server should only occur if the processing that will be repeated is idempotent.

It's impossible for a remote stub to know which of the three points of execution that the request was in when the failure occurred, even though failures of requests that have not even begun method execution, since fail can happen at any of the three points above.

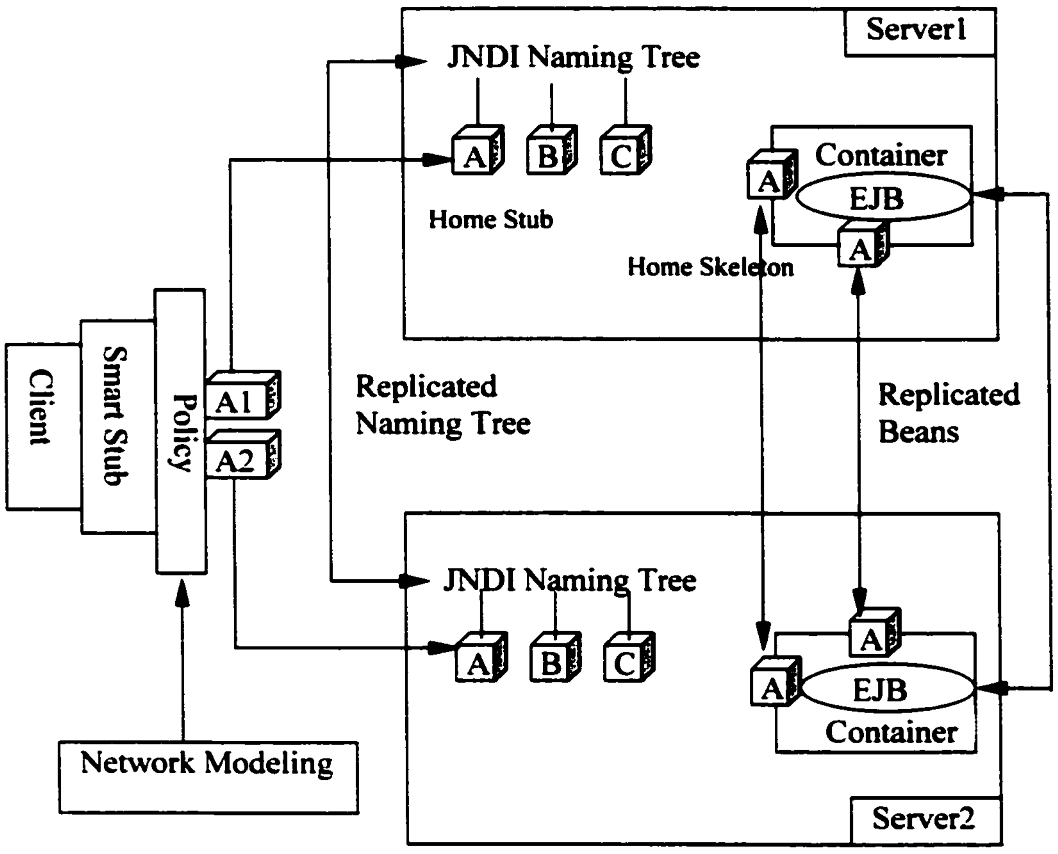


Figure 4.7 EJB Clustering

Remote stubs can only automatically fail over requests that were sent to methods that are idempotent. The EJB methods can be coded as idempotent or non-idempotent by developers. Idempotency of methods can be applied to all types of EJBs. Automatic fail over of non-idempotent methods is only possible for requests that failed before the server-side object began execution. However, since it is generally impossible to detect at which point a request failure occurred, it's impossible to provide automatic fail over of non-idempotent methods. Fail over of non-idempotent methods must occur programmatically by the client that originated the request.

An EJB can be clustered in three locations: the container, the home stub, and the remote stub. The container resides within the application server while the remote and home stubs are generated by the application server or a compilation tool and downloaded to run remotely on a client. Different cluster-specific logic can be applied to these locations.

The remote stubs are downloaded by Java clients and run local on a remote machine. Also, since remote stubs are generated at runtime by an EJB container or at build time by an EJB compiler. Container Provider (CP) can develop load balancing and fail over algorithms for EJBs that does not execute in their own application server. CPs can generate logic that is incorporated as part of a remote home or remote stub that executes before or after each invocation on the stub on the remote client. This, along with the logic that can be incorporated directly into a container on an application server, provides a large permutation of options for clustering logic and EJBs. A particular CP decides the level, type, and occurrence of cluster logic.

4.5.1 Container – An Application Server Vendor (ASV) can provide some load balancing and fail over logic directly within the container. If an EJB were deployed to multiple servers in a cluster, each server in the cluster would have a container hosting that EJB. The containers could communicate with one another using an inter-server communication protocol [These01]. The containers could use this inter-server communication protocol to re-route requests, replicate stateful data, or communicate actions. For example, if a container on one server is burdened with requests for a particular type of stateful session bean, the container may be able to forward the request to its counterpart located in another server. When a stateful session bean is created, a backup copy can be placed on another server in the same cluster. The backup copy is not used unless the primary fails at which point the backup becomes the primary and nominates another backup. Every time a transaction on a stateful session bean commits, the bean is synchronized with its backup to ensure that both locations are current. If the container ever has a system failure and loses the primary bean instance, the remote stub of the bean fails over invocations to the server that contains the secondary instance. The inter-server communication protocol would be used in this scenario to replicate the stateful data and to communicate notifications about the state of the bean to each container. Stateful session bean replication is a highly desirable feature since it provides non-interruptible access to some business logic. It is not designed, however, to provide persistent access to data since simultaneous failure of the primary and backup servers cannot be recovered. Entity beans should always manage persistent data.

4.5.2 Home Stub – This object is the first object accessed by remote clients and runs locally on a client's virtual machine. In the most common scenario, remote stubs are generated at deployment time by an EJB compiler that is provided by the ASV [These01]. Since stub code is auto-generated by a ASV, the underlying logic in a stub can be ASV-specific. ASVs can instrument method-level load balancing and failover schemes directly in a stub. Since the primary responsibility of the home stub is to direct create or load requests to a skeleton that can service them the server which handles a request is not important. Effectively, every create(), find(), and home() method invocation can have its request load balanced to a different home skeleton in the cluster.

The most common scenario for home stub generation is the use of a ASV-specific EJB compiler. An EJB compiler will generate the compiled Java classes that make up the stubs, skeletons, and any other ASV-specific implementation files needed to support the EJB and place them back into the JAR file. Some application servers can use interception technology such as the JDK 1.3 Proxy class to automatically generate remote home and remote stub logic dynamically at runtime. This is also called hot deployment.

Whether or not an application server uses interception technology or creates custom classes for the stubs and skeletons does not alter the places where cluster-based logic can be inserted.

4.5.3 Remote Stub – Remote stub is instantiated by the home skeleton and returned back to the client. This object can perform the same types of load balancing and failover that a home stub can do. A remote stub that references an EJB instance with stateful data that may or may not be replicated to other servers needs to be intelligent when it forwards invocations. Remote stubs must load balance and fail over requests to instances that can properly handle the request without disrupting the system.

The load balancing and fail over logic either existed in a remote home or remote stub or in the container itself. This is an indirect implication of the fact that load balancing and fail over logic reside outside of the system that is being clustered. In regards to EJBs, only clients that make use of remote home or remote stubs will be able to see load balancing or fail over of requests [These01].

4.6 Summary

Overall. we can see from above that there is a vast array of configurations that clusterable EJB may be conformed to.

A remote client initially connects to a cluster. The client can be stand alone Java program using RMI/IIOP. a servlet operating with WAS, or another EJB. The client does so by using the JNDI library. The client issues the request using a cluster address. A load balancer, proxy. or other piece of Initial Access Logic intercepts the first request and can perform a load balance to one of the nodes in the cluster.

The client receives the *InitialContext* and executes a *lookup()* method invocation to receive a remote home stub. If the naming servers are replicated in the cluster, all of the naming servers will have the home stubs for all EJBs in the cluster, the *InitialContext* object can load balance the *lookup()* invocation to any node. The client receives the remote home stub and executes a *create()*. *find()* or *home()* method.

If the same EJB container is hosted on multiple servers in the cluster, the remote home stub can forward the request to any of the containers on any of the servers using a load balancing algorithm. Since *create()*, *find()*, and *home()* invocations are stateless invocations that do not need to be pinned to a particular instance (container), a remote home stub can load balance its requests.

The client receives a remote stub in response to a *create()* or *find()* method invocation. If the remote stub is not pinned to a particular instance, then any business operation made on the remote stub can also load balance the request between servers in the cluster.

The WAS Vendor and application developer can have many choices for implementing of workload balancing and fail over through smart stub, caching, container, database and specific programming.

Currently, many ASVs provide powerful WAS, such as BEA WebLogic, IBM WebSphere, Oracle9iAS, SUN iPlanet, etc. these servers are not only support J2EE specification, but also support clustering WASs. Different clustering protocols are used by different ASVs. For example, The WebLogic service advertisement protocol (WSAP) is based on IP multicasting and it transmits three types of messages: heartbeats, announcements, and state dumps.

Further complex protocol that is based on the network monitoring information can be developed. One possible approach is using Queuing Theory for workload balance.

4.7 EJB Future Development

The followings are possible EJB development in future:

4.7.1 Functional Separation

Currently EJB container, servlet and JSP engine are in the same WAS instance, they benefit from being in the same memory space. It is better to separate them to different container according to their usage. Since EJB may use much more than the others, or vice versa. For example, servlets are configured in 5 WAS instances and EJBs are configured in 10 WAS instances. They are all full utilizing servers processing power. If they run on different computers, there is a price to be paid by having the requests travel over the network. In some case the price of the network is worthwhile because of the scalability gained from having the load distributed over various machines.

4.7.2 Firewall Restrictions

The boundaries between physical hardware/software layers provide potential points for defining the web application's De-Militarized Zone (DMZ). However, not all boundaries can support a physical firewall, and certain boundaries can support only a subset of typical firewall policies. For example, it's difficult to achieve that placing firewall with full functionality between distributed servers while still achieve good load balancing and fail over.

4.7.3 Dynamic EJB Generation Proxy

When we create a new EJB, it has to be deployed in the deployment descriptor. With the DynamicProxy API we can generate such an EJBean proxy but we cannot control the name of its class nor enter it in the deployment descriptor instead of the original EJBean. The client has to refer to (get the home from) the EJBean proxy instead of the original. It should not against the EJB separation between deployment and runtime phases.

DynamicProxy in Java is very elegant to implement client side and server side interceptors or any proxy design patterns. But the architecture of EJB is more static and force to declare the "servers" (EJBean) before runtime.

EJB 2.0 states that only entity beans that implements local interfaces (javax.ejb.EJBLocalObject and javax.ejb.EJBLocalHome) can participate in an ejb-relationship. This means that you would have to have two sets of entity EJBs. One set for EJB 1.1 and a second set for EJB 2.0. Unfortunately, all methods in the remote component interface MUST throw java.rmi.RemoteException. All methods in the local component interface MUST NOT throw java.rmi.RemoteException. At a glance it could seem that it would mean two sets of session EJBs. If your session beans use entity beans directly, i.e. session beans import interfaces of entity beans; the quick and dirty route would be to have two sets of session EJBs as well. But this is far from ideal regarding reuse, maintenance and portability.

4.7.4 Transaction and Security Context Definition

The problem of the state associated with every method call, such as the transaction context and security identity of the caller. How to move this state from the client EJB to the server EJB is not defined in the Enterprise JavaBean 1.1 specification or the RMI specification. This lack of run time compatibility between EJB containers is considered a weakness of the current situation. [Subra00]. The EJB2.0 specification does specify RMI/IIOP and other CORBA protocols as the solution to the interoperability problem. However, the 2.0 specifications made transaction interoperability and certain aspects of security interoperability optional. In other word, the situation is still not completely resolved.

4.7.5 Multiple Developer Support

Multiple developers operate EJBs from a central EJB container, yet be totally independent of each other, and hence no impact of one developer having to re-start their EJB context to allow redeployment of a JAR.

In this chapter, we discussed various strategies in dealing with fail over and workload balancing. Many of them are embedded in WAS, such as WebLogic and WebSphere. These embedded functions are convenient for developers. In the next chapter, we will use WebLogic for multi-server model design approach.

Chapter 5 Prototype Design and Implementation

The aim of this thesis is mainly to study and describe using EJB for building multi-server design model. and to investigate it in practice by using it in an appropriate application that requires that kind of services provided by EJB. Earlier we talked about several features and properties of the EJB. such as. scalability, portability and its being component-based. These issues must be fulfilled by the EJB server. Still. EJB servers are supposed to provide other services that are known to give EJB its strength: transaction management. persistence and security. This system shall be able to handle multiple distributed clients concurrently access. The system can be configured as multiple physical servers. More over. this prototype will give the model that has sufficient functionalities and capacities to be useful in real world. The application described below has been chosen with these services in mind.

The application is an on-line code service and shopping system. First, the system brings the users to its homepage. After check the user validation, it will show a list of codes. remark and prices for the codes. Also. the system can generate the code according to user's option. We use some tools such as WebSphere Studio. JBuilder and UML in our prototype design.

5.1 System Architecture

We are trying to make thin client approach as we discuss in the previous chapter. The client side only needs a Web browser and JVM. The server side provides all the services. The system architecture is shown in Figure 5.1 and Figure 5.2. Users start from Index.jsp. This JSP page will automatically compiled to a servlet class before its first use. The Index.jsp starts a session. This session will be maintained until the browser is closed or a timeout occur. The value of timeout can be defined in the WebLogic property file:
Weblogic.httpd.session.timeoutSecs=integer

We built the class and unit library. The JSP files calls EJBs that mapping to the database tables. This library is for code generation. Figure 5.1 shows these relations.

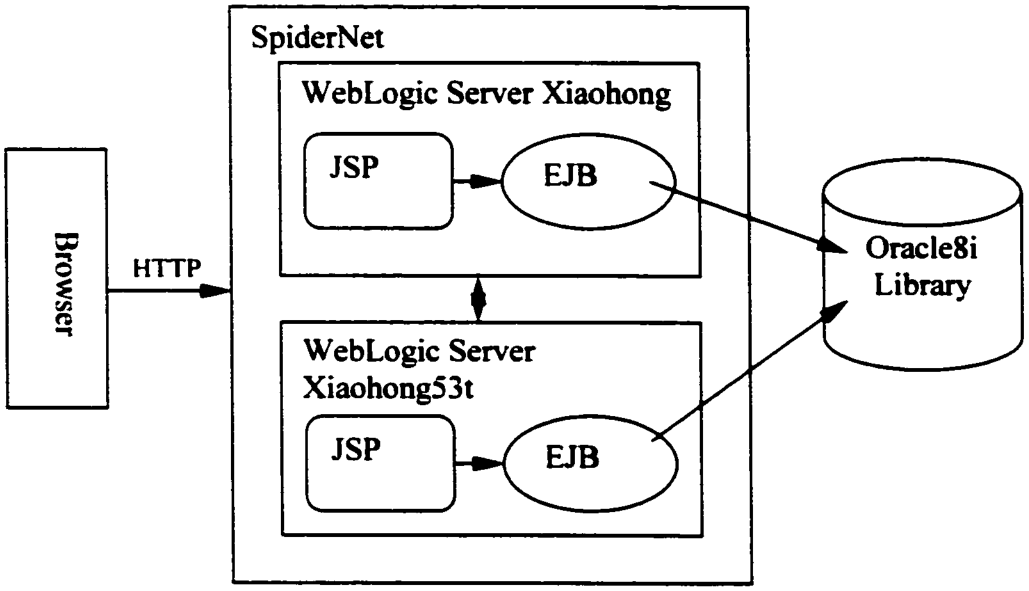


Figure 5.1 Clustered Servers in SpiderNet

Since we programmed several different types of files in our system, for easy identification, we define the following naming conventions shown in table 5.1:

Class Type	Suffix	Example Files
Servlet	S	CodeSearchS, UserSearchS
Stateless session bean	SL	CodeServiceSL,
Stateful session bean	SF	OrderSF
Entity bean	EB	UnitEB

Table 5.1 Naming Conventions

Figure 5.2 shows system function modules.

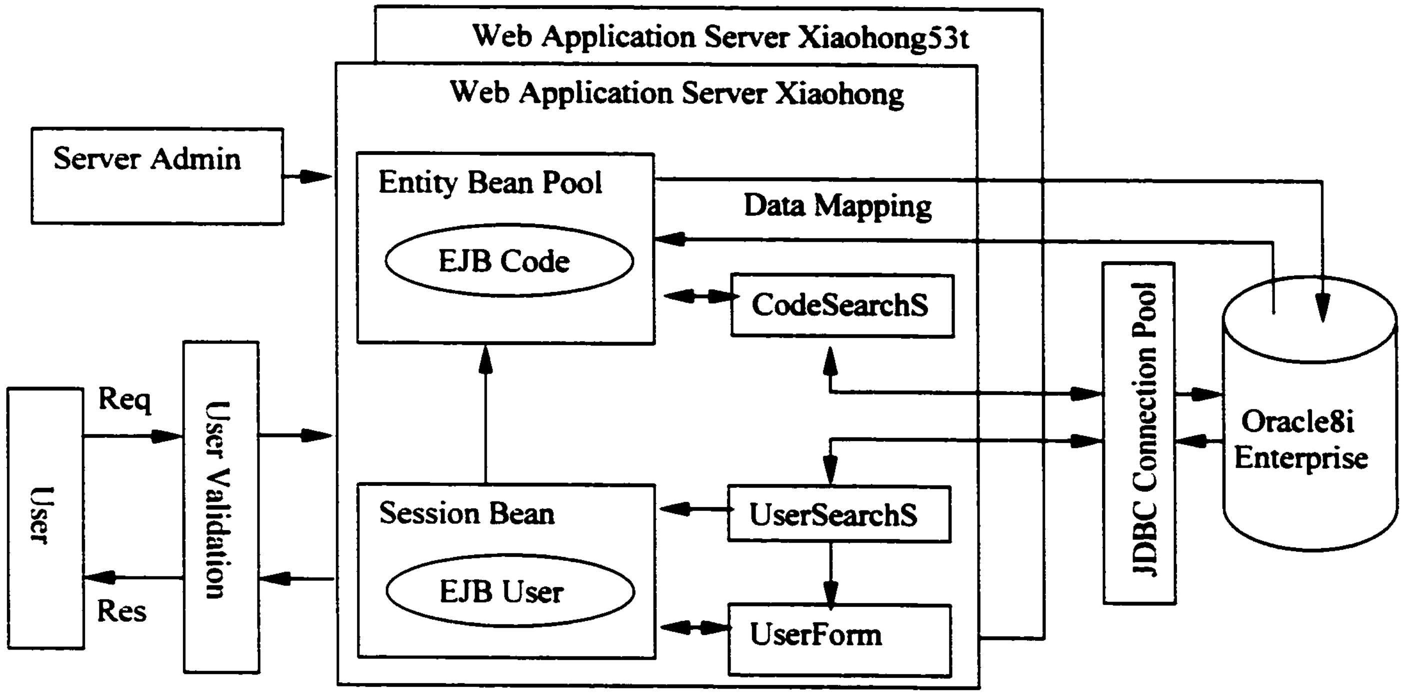


Figure 5.2 System Function Modules

These functional modules are explained in the followings:

User – User access SpiderNet by Internet browser

User Validation – Check the user list to see whether he/she is on the list.

Server Admin – Administrator can configure the properties of WebLogic. He/she should have system administrator privilege.

Customer Search – This module gets data from user and sends it to UserForm.

Session Bean – Temporary store the user information. It has transaction properties. Since multi-clients connect to the server concurrently, each user has his own session. Further, when the user is not active for a period of time, (it can be set in Weblogic property file) the session bean can be passivated to database. The object can be used for another user session. The bean pools size can be set in the EJB description file.

Code – This is an entity bean. This entity bean maps to the database table Code and stores the code information. Accessing to the database may be expensive and takes time. The database connection building up may up to 1 second. It's possible to improve performance by caching the database data at the server level. In such a case, if multiple consecutive requests are directed to the same server, they may find the required data still in the cache, and thereby reduce the overhead of access to the underlying database.
Another advantage is when many users retrieve the same code from the system, the system do not need to create database connection for each user. The system can create multiple Code objects instead. This saves database connection time and database connection resource, since Oracle8i can not support a large number of connections.
JDBC Connection Pool – a database connection resources that can be shared for all the connected users. This property can be set in Oracle8i and WebLogic property file. The value set in WebLogic file should not more than the value set in Oracle8i. It increases the efficiency of using database, since Oracle8i enterprise supports more than 100 concurrent connections. We will further discuss it in section 5.4.

Although the system is not complicated, many concepts we discuss earlier are practiced. For example, CodeService is a stateless bean, OrderSF is a stateful bean and Code is an entity bean.

5.2 Development Environment

In order to develop an application that is multi-server based and J2EE environment supported, first we need to setup the development environment. We need powerful machines. Two computers, Xiaohong and xiaohong53t, are networked in this prototype design. They are configured as following:

Xiaohong: PIII 550 with 160M memory

Operating system: Windows 2000 server

Software:

C:\weblogic	BEA WebLogic Version 5.1
C:\jdk1.3	Sun JDK Version1.3.0.2
C:\oracle\ora81	Oracle8i Version 8.1.6

C:\spidercode	SpiderNet Code

Xiaohong53t: PII 350 with 256M memory
Operating system: Windows NT work station 4.0 with SP6a
Software:

C:\weblogic	BEA WebLogic Version 5.1
C:\jdk1.3	Sun JDK Version1.3.0.2
C:\spidercode	SpiderNet Code

The directories are configurable. Other software such as IE 5.0, GNUmake, JBuilder 4.0, WebSphere Studio, etc are installed on both computers for easy development and debugging.

The computers are networked through a hub: PUREDATA PDC 8023U-4P. The network card for PII 350 is PUREDATA 10/100 Ethernet and for PIII 550 is Linksys 10/100 USB network card. Since it's USB interface, the fastest speed is 12Mb/s in theory. In practice, the speed is much lower than 12Mb/s.

5.3 Multi-Server Configuration

Since WebLogic 5.1 supports multi-server clustering configuration, we can use this property instead of programming our won. The configuration include the following steps:

1. Configure DNS. Assign an IP address to each WebLogic Server and a multicast address for the cluster.
2. Set up weblogic.properties files for the cluster.
3. Locate classes and register classes in the properties files.
4. Specify deployment and startup class properties in the properties files.
5. Set the CLASSPATH for each WebLogic host.

The two networked computers are assigned the following IP addresses:

```
Xiaohong: 192.168.0.1          #mycluster Yu
```

Xiaohong53t: 192.168.0.7 #mycluster Yu

A server name for each server in the cluster points to the IP address for each server. The cluster name is Yu. These IP address are reserved and not used on the Internet. The servers in a WebLogic Cluster use IP multicast to communicate between themselves. This is set in the weblogic.properties file for each Server.

5.3.1 Default and global properties

The global properties are set in the following file for both computers:

/weblogic/weblogic.properties

These properties includes:

System password = xxxxxxxx

All servlet registrations. with the weblogic.httpd.register property.

5.3.2 Cluster properties

Cluster properties is configured in the file:

/weblogic/yu/weblogic.properties

All clusterable deployments. registrations of startup classes are registered with the property weblogic.system.startupClass. All paths that we configure in the cluster properties file. such as servlet classpath, document root, etc.. should point to the correct directory.

5.3.3 Per-server properties

/weblogic/<mycluster>/serverxxx/weblogic.properties

RMI objects that have been compiled without the `-clusterable` flag

weblogic.httpd.clustering.enable=true

weblogic.cluster.defaultLoadAlgorithm

weblogic.httpd.session.persistence=true

weblogic.httpd.session.persistentStoreType=replicated

The first property is obvious. We just simply set cluster mode to true for clustering configuration. The load balancing algorithm set the load balancing strategy to be used between replicated services. The default is "round-robin", with acceptable values of "random". "round-robin". and "weight-based". Load balancing is explained in previous chapter.

The other property settings are the same as single WebLogic server setting. Some of them are:

```
weblogic.system.listenPort=7001
weblogic.password.system=weblogic
weblogic.allow.execute.weblogic.servlet=everyone
weblogic.security.ssl.enable=true
weblogic.system.SSLListenPort=7002
weblogic.security.certificate.server=democert.pem
weblogic.security.key.server=demokey.pem
weblogic.security.certificate.authority=ca.pem
weblogic.httpd.session.enable=true
weblogic.httpd.http.keepAliveSecs=60
weblogic.httpd.https.keepAliveSecs=120
weblogic.jdbc.enableLogFile=false
weblogic.jdbc.logFileName=jdbc.log
weblogic.httpd.initArgs.file=defaultFilename=index.html
weblogic.httpd.register.*.shtml=weblogic.servlet.ServerSideIncludeServlet
weblogic.httpd.register.proxy=weblogic.t3.srvr.HttpProxyServlet
weblogic.httpd.servlet.classpath=/weblogic/myserver/servletclasses
weblogic.httpd.servlet.reloadCheckSecs=1
```

The default port for WebLogic is 7001, although we can set the other port. Same thing to the security listen port, it is set to 7002. The security keys are stored in democert.pem, demokey.pem and ca.pem file. The session keep alive is set to 60 second. If the session is not active for this period, it will be closed and the resource will be free for creating other sessions.

5.4 JDBC Connection Pools

Establishing a connection to a database is a pretty expensive operation in the sense that it can take up to a second. Because of this, the traditional way of handling database connections from a client is to establish a connection between the client and database when the client application starts and then maintain the connection until the client ends. This architecture is costly because it needs as many connections as active clients, and worse still, the connection is not being used efficiently, as the interaction with the database is intermittent. And the connection is almost always idle. A more efficient alternative is the use of connection pools. The idea is that a predefined number of database connection are established initially and then made available to the clients when needed. As the connection has already been established, when the client requests one it is available immediately. When the client is done with the connection, it is not dropped but returned to the pool and made available to other clients.

In this prototype design, a JDBC connection pool yuconpl was created. It is defined as the followings:

```
Weblogic.jdbc.connectionPool.yuconpl =\
        url=jdbc:weblogic:oracle, \
        driver=weblogic.jdbc.oci.Driver, \
        loginDelaySecs=1, \
        initialCapacity=1, \
        maxCapacity=20, \
        allowShringking=true, \
        shrinkPeriodMins=10, \
        refreshTestMinutes=10, \
        testable=dual, \
        props=user=scott; password=tiger; server=Oracle8i
```

loginDelaySecs define the number of seconds to wait between each attempt to open a connection to the database. This is to simulate the practical situation, because not all clients build the database connection at the same time.

InitialCapacity defines the number of connections that have to be established when the WebLogic server is starting.

MaxCapacity defines the maximum number of connections a pool can have. CapacityIncrement means that when all connections are in use and an additional one is requested, WebLogic will automatically increment the number of connections.

ShrinkPeriodMins, the connection pool can shrink after this time periods, if the allowshrinking is set to true.

RefreshTestMinutes is for testing the database connections are still alive. If the test fails, a new database connection will be established.

5.5 System Features

This application is running as a Website. Multiple servers are running for code service support. Not only the system supports multiple client concurrent connections, but also client requests can be distributed between servers, making the server group appear as a single logical server to the client. Increasing the number of servers improves the server's process ability and availability. On this configuration model, Cluster servers can enter and leave a cluster at any moment.

The system provides the following features:

- **User Registry:** New user can register and old user has his own records. The user information can be updated.
- **Code List:** The system shows the available codes to users.
- **On-line Shopping:** Users can buy the code that he need from the SpiderNet Web site.
- **Code Generation:** users can select the features from options and generate customized code that they want.

- **Billing System:** The system can display the prices of the codes, add the total cost and charge the user who buy the codes
- **Security:** The information between users and system, such as credit card number and codes, are exchanged through SSL security protocol.

5.6 Design Implementation

First we need to setup database scheme and create database tables. Then create the JSP files and Java class files. The source codes and executable java class files are stored in src and srv directory respectively. The flow diagram is shown in Figure 5.3

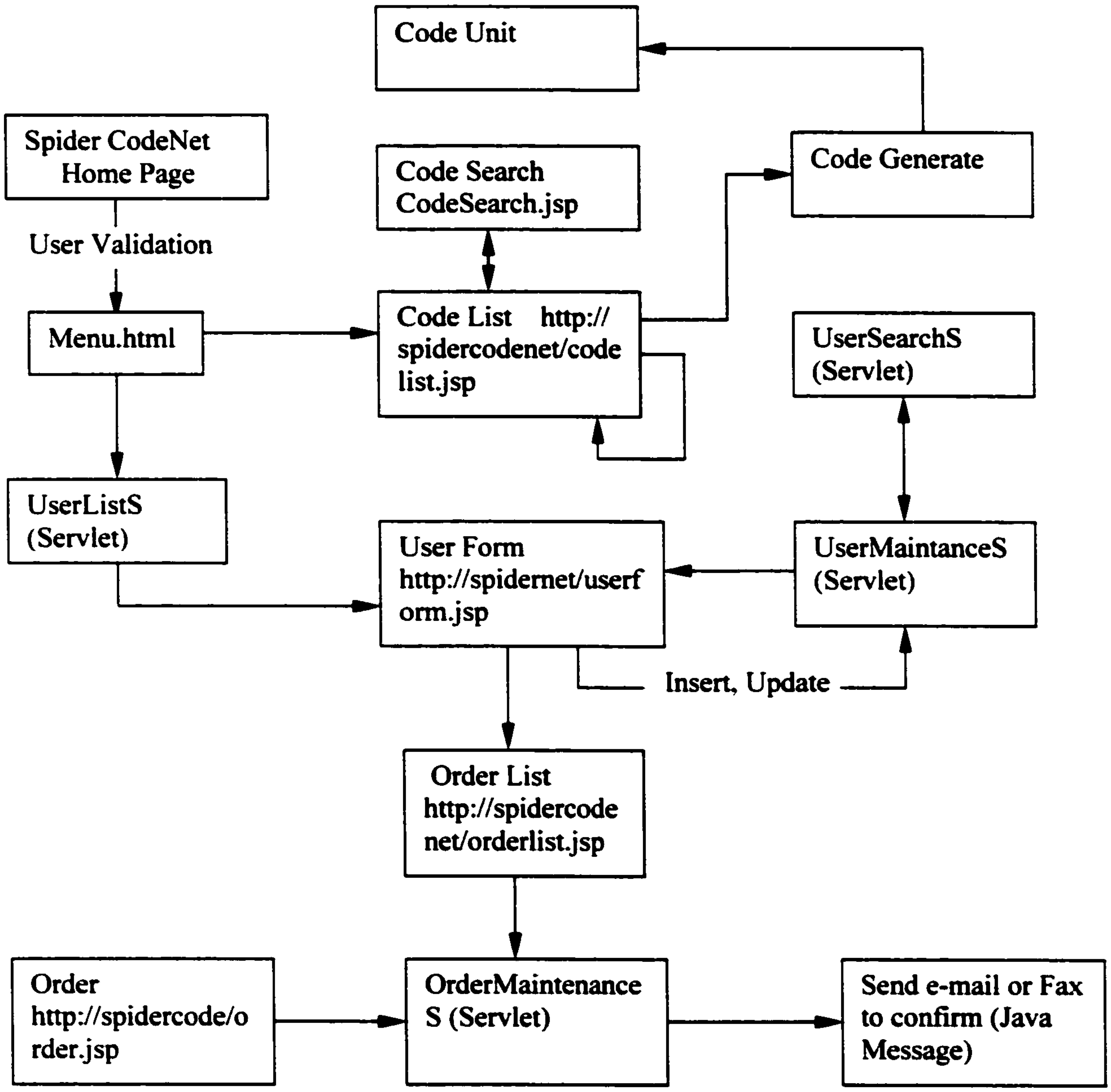

Code Unit
Spider CodeNet Home Page
Code Search CodeSearch.jsp
Code Generate
User Validation
Menu.html
Code List http://spidercodenet/codelist.jsp
UserSearchS (Servlet)
UserListS (Servlet)
User Form http://spidernet/userform.jsp
UserMaintanceS (Servlet)
Insert, Update
Order List http://spidercodenet/orderlist.jsp
Order http://spidercode/order.jsp
OrderMaintenanceS (Servlet)
Send e-mail or Fax to confirm (Java Message)

Figure 5.3 System Flow Chart

The sequence diagram for user session is illustrated in Figure 5.4. Sequence diagram for user search is shown in Figure 5.5. When the customer has selected the desired items and added to his/her order list. The system calculates and shows the total price of the code.

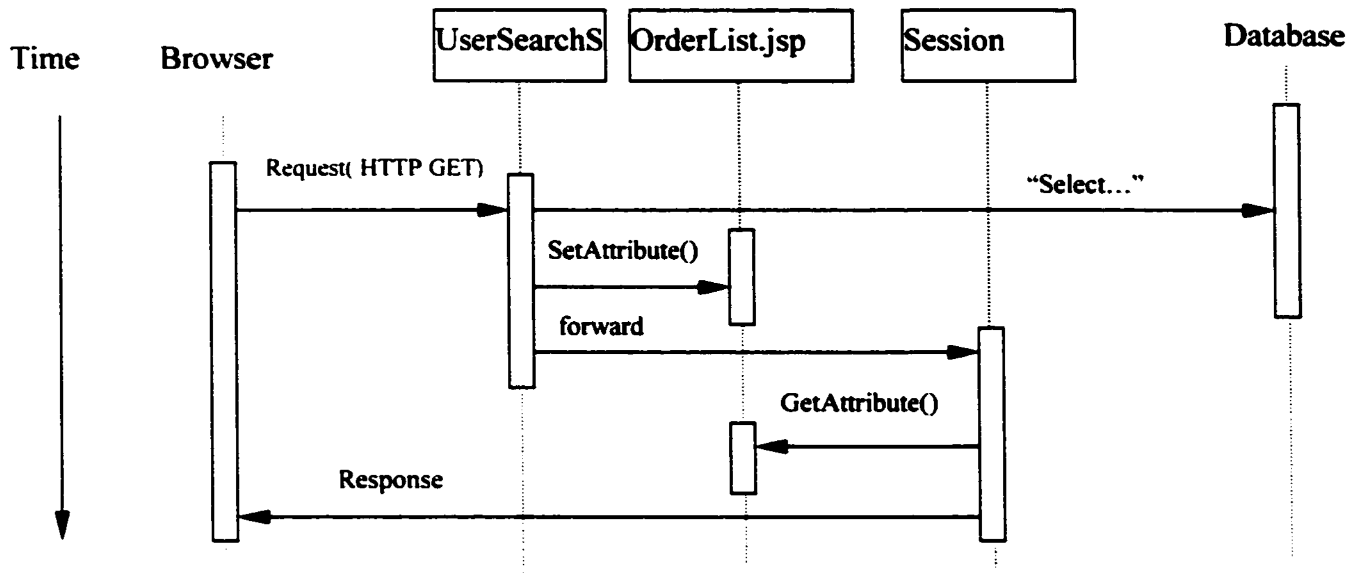


Figure 5.4 Sequence Diagram for User Session

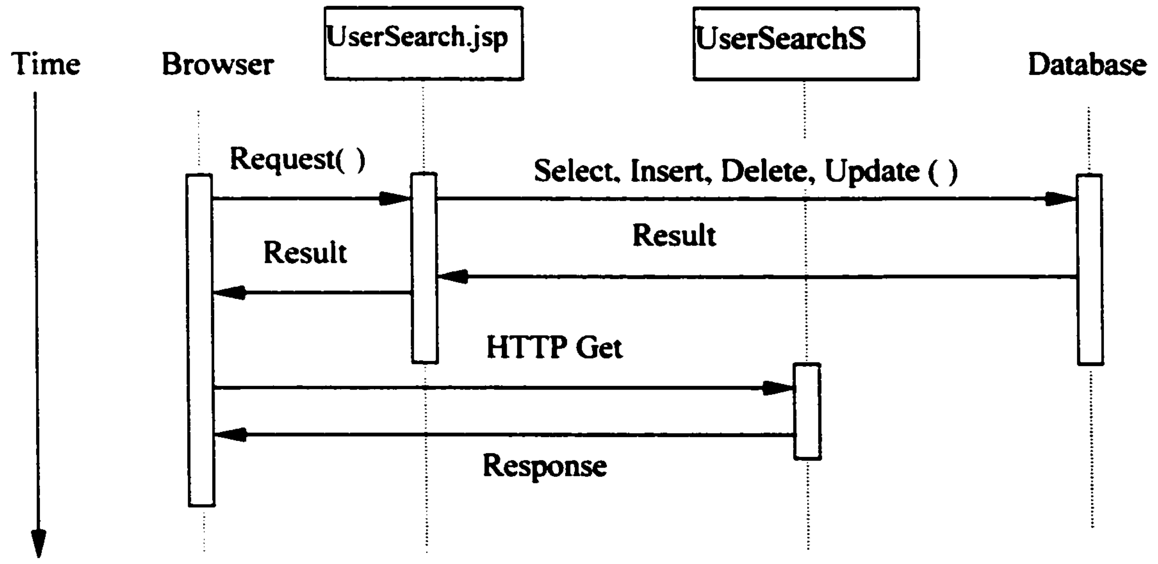


Figure 5.5 Sequence Diagram for User Search

When the client wants to generate the code, the system calls the codeGenerate. The sequence diagram for code generation is shown in Figure 5.6

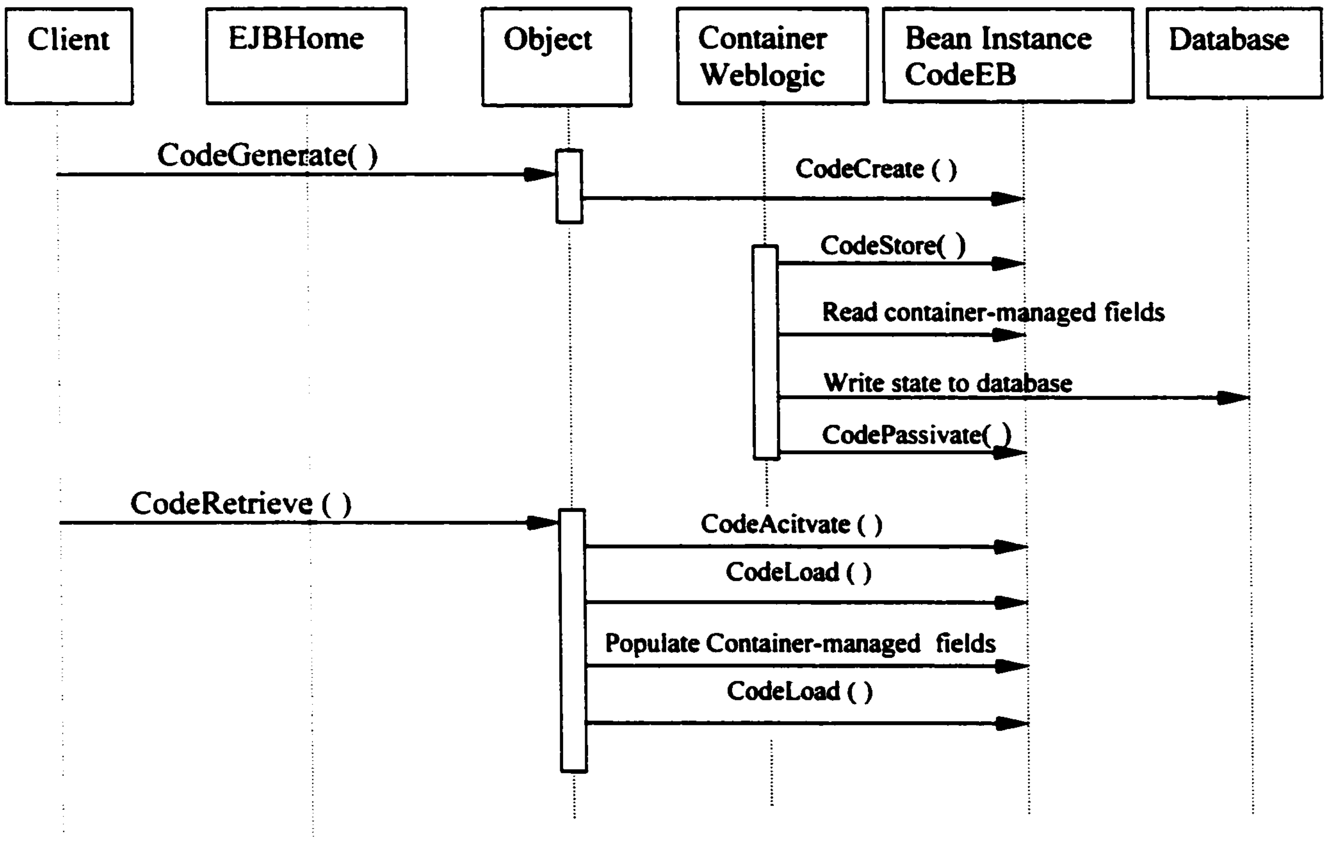


Figure 5.6 Sequence Diagram for Code Generation

The class generation is achieved by retrieve units from the Unit table, which compose method library. Figure 5.7 shows the sequence diagram of this operation. Figure 5.8 shows the relations of UnitEB, Code and Unit tables.

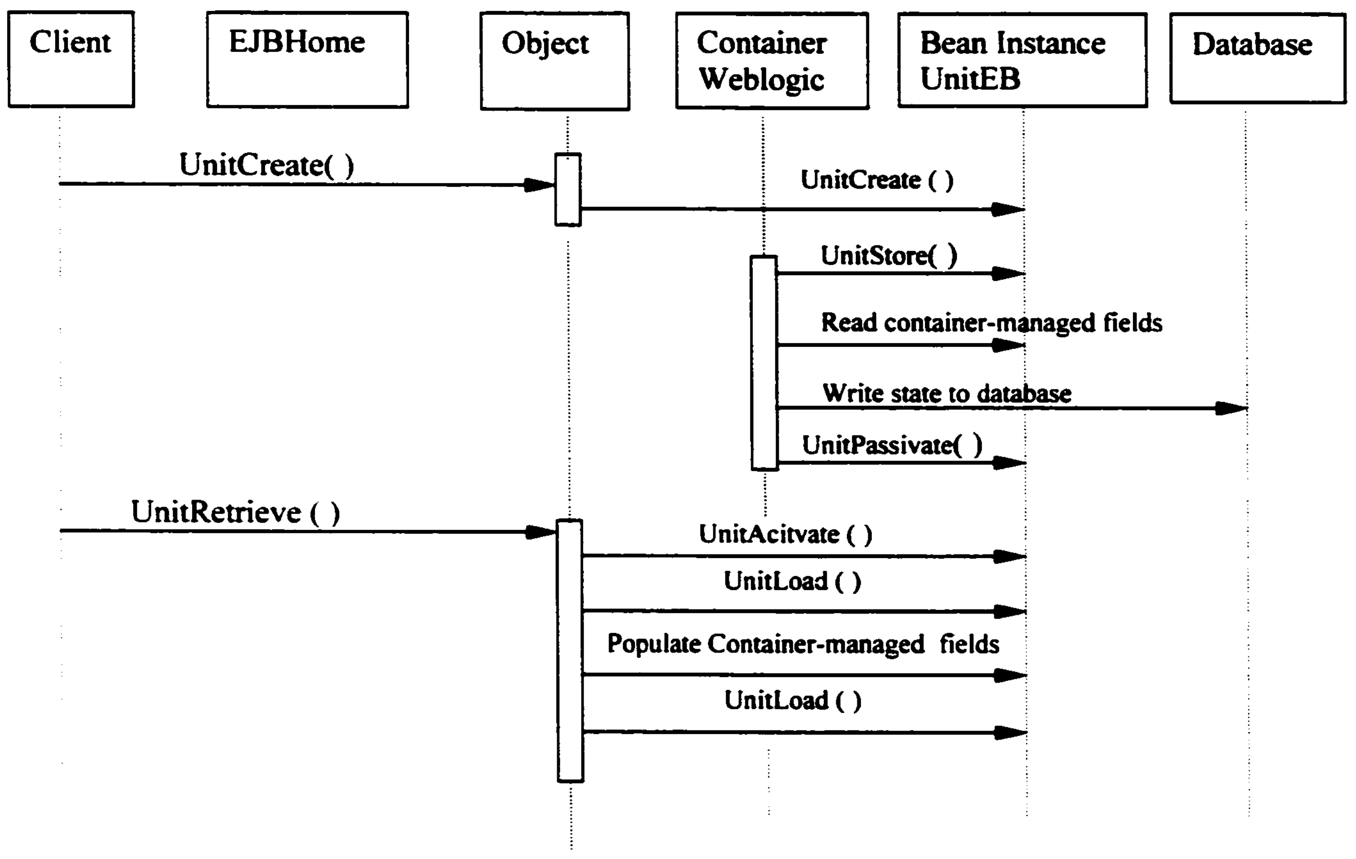


Figure 5.7 Sequence Diagram for Unit Generate

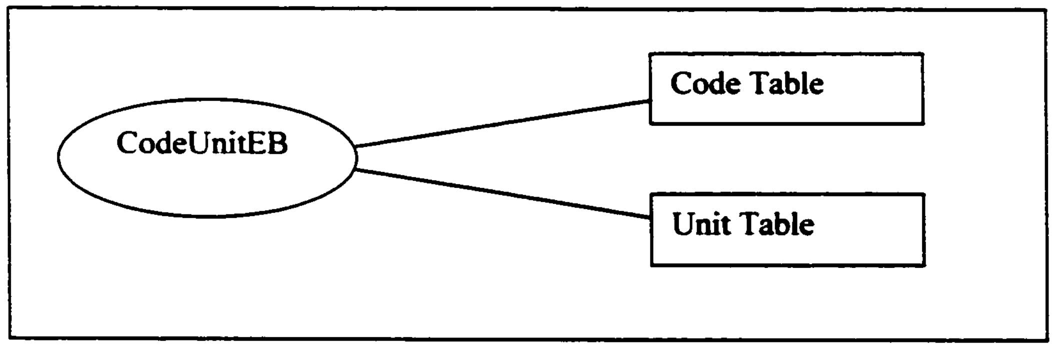


Figure 5.8

5.7 EJB Classes

6 EJB are designed in the SpiderNet System:

- Stateless Bean: CodeService, OrderService, Service
- Stateful Bean: Order
- Entity Bean: Code, Unit

The following are the EJBs classes:

5.7.1 CodeServiceBean:

Bean Class: CodeServiceBean

Home Interface: CodeServiceHome

Remote Interface: CodeService

5.7.2 OrderServiceBean:

Bean Class: OrderServiceBean

Home Interface: OrderServiceHome

Remote Interface: OrderService

5.7.3 OrderBean:

Bean Class: OrderBean

Home Interface: OrderHome

Remote Interface: Order

5.7.4 ServiceBean:

Bean Class: ServiceSLBean

Home Interface: ServiceSLHome

Remote Interface: ServiceSL

5.7.5 CodeBean:

Bean Class: CodeBean

Home Interface: CodeHome

Remote Interface: Code

Primary Key: compound of (C_Name, C_Size) from Code table

5.7.6 UnitBean:

Bean Class: UnitBean

Home Interface: UnitHome

Remote Interface: Unit

Primary Key: compound of (U_C_Name, U_C_Size, U_M_Name) from Unit table

These EJBs are called by the relative JSP files or other EJBs. Each EJB has deployment description files. The following are description file for EJB CodeServicesSL:

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
   <session>
       <ejb-name>xiaohong.yu.CodeServicesSL</ejb-name>
       <home>xiaohong.yu.CodeServicesSLHome</home>
       <remote>xiaohong.yu.CodeServicesSL</remote>
       <ejb-class>xiaohong.yu.CodeServicesSLBean</ejb-class>
       <session-type>Stateless</session-type>
       <transaction-type>Container</transaction-type>
   </session>
  </enterprise-beans>
  <assembly-descriptor>
   <container-transaction>
       <method>
        <ejb-name>xiaohong.yu.CodeServicesSL</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
       </method>
       <trans-attribute>Required</trans-attribute>
   </container-transaction>
  </assembly-descriptor>
 </ejb-jar>
```

Deployment descriptor weblogic-ejb-jar:

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
   <ejb-name>xiaohong.yu.CodeServicesSL</ejb-name>
   <caching-descriptor>
       <max-beans-in-free-pool>100</max-beans-in-free-pool>
   </caching-descriptor>
   <enable-call-by-reference>false</enable-call-by-reference>
   <jndi-name>xiaohong.yu.CodeServicesSL</jndi-name>
  </weblogic-enterprise-bean>
 </weblogic-ejb-jar>
```

Similar. we have description files for OrderServicesSL, OrderSF, ServicesSL, CodeEB and UnitEB. The contents of these files can be found in Appendix A.

5.8 Security

There are two kinds of security check in SpiderNet:

1. WebLogic security check
2. Register.jsp

WebLogic can set local user ID and password in its property file:

```
Weblogic.password.xiaohong=xxxxxxxx
```

System can use register.jsp for security check by cookie or URL rewriting.

Add the following properties to weblogic property file for security:

```
Weblogic.security.ssl.enable=true
Weblogic.system.SSLListenPort=7002
Weblogic.security.certificate.server=democert.pem
Weblogic.security.key.server=demokey.pem
Weblogic.security.certificate.authority=ca.pem
```

For remote user. there is one or more firewall between client and server. Client can choose one way or two-way SSL security option. Client needs to install the certificate authority key in its browser if he chooses two-way security option. This security check mechanism can also apply to local users.

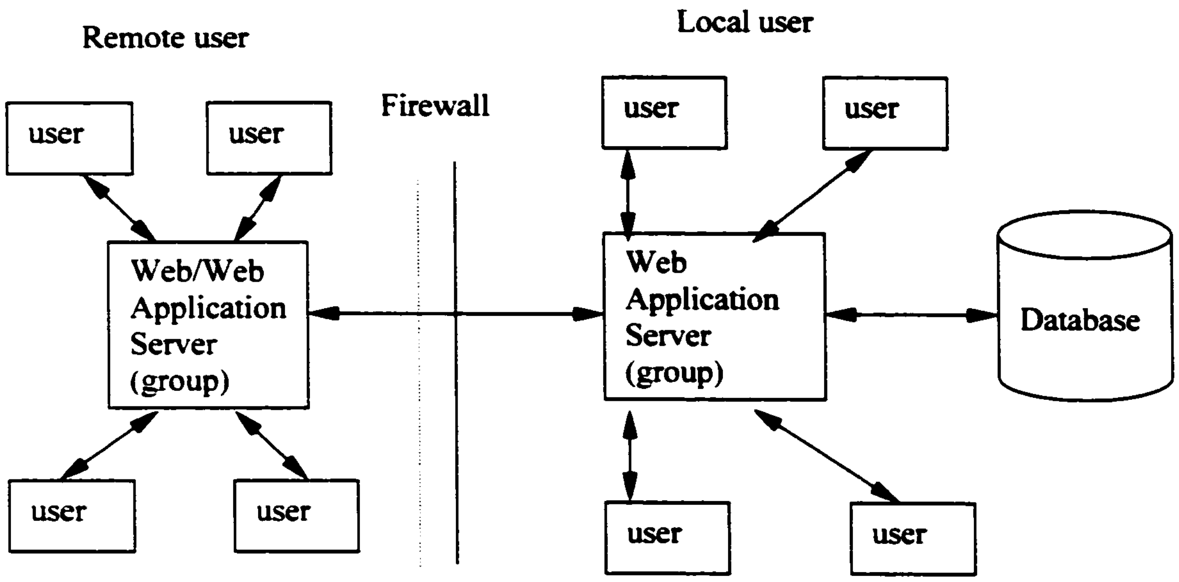
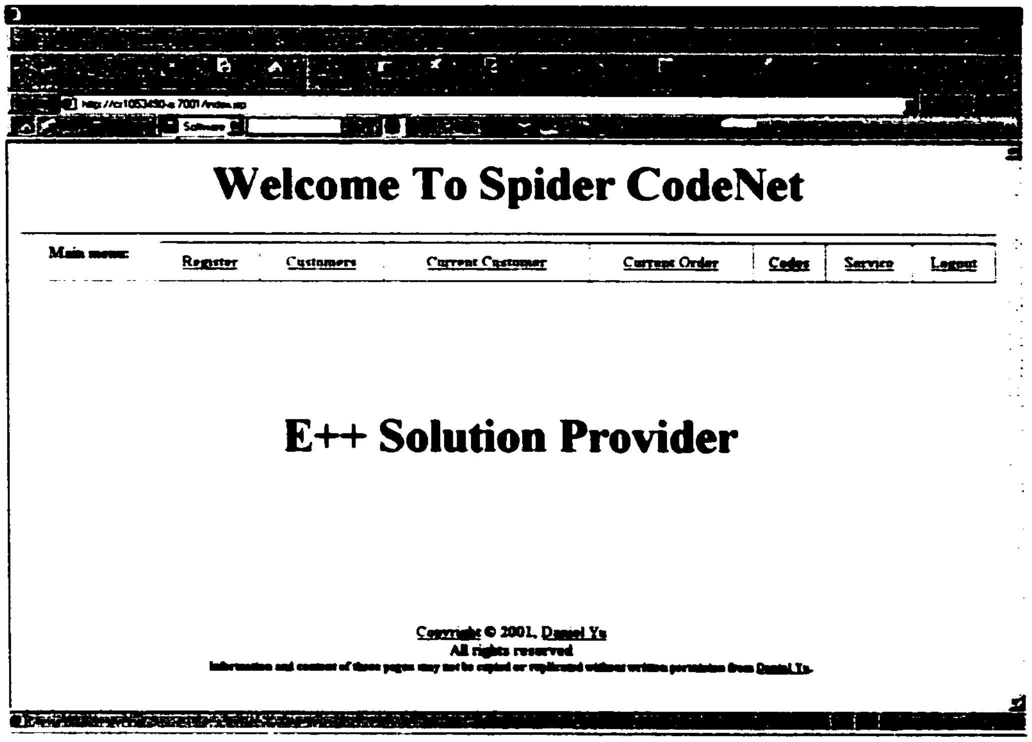


Figure 5.9 Users and Security

5.9 System Operations

Servers are started by going to directory "SpiderCode", setting runtime environment and inputting the command "SpiderRun". This will start WebLogic servers. The servers will output lots of run time messages. The system interface is presented to its customers through a Web site and a customer interacts with the system using a Web browser. At first. a user will locate to the SpiderNet homepage by pointing the RUL to http://xiaohong:7001/index.jsp or http://xiaohong53t:7001/index.jsp for Xiaohong and Xiaohong53t respectively. as is shown in Figure 5.10. From the main menu, the user can go to register. search codes. customers, etc.

Since it is presented as a web site. it is self-illustrated and straightforward to use. It is not necessary to cover every detail of usage of the system. The following Figures show some screen shots of the system during run time. We brievely introduce them.

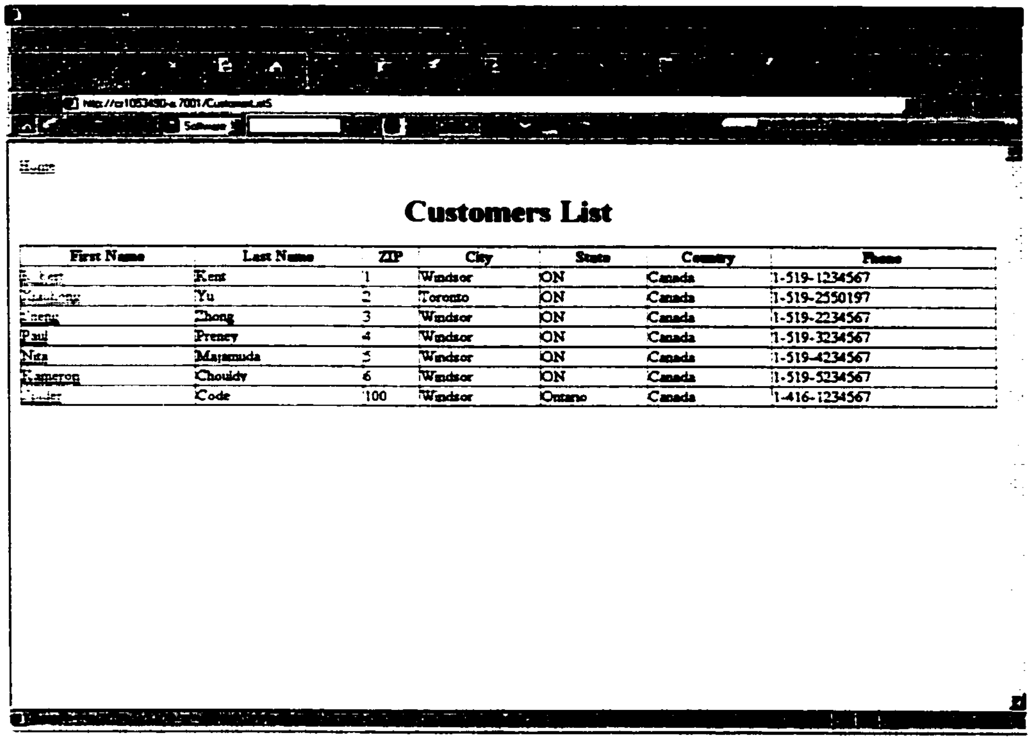

Welcome To Spider CodeNet

E++ Solution Provider

Copyright © 2001, Daniel Yu
All rights reserved
Information and content of these pages may not be copied or replicated without written permission from Daniel Yu.

Figure 5.10 SpiderNet Home Page

Input user phone number for the old user and system retrieves the user information by simply click the username. New users need to register first. Figure 5.11 is the customer list and Figure 5.12 is the new user registration form.

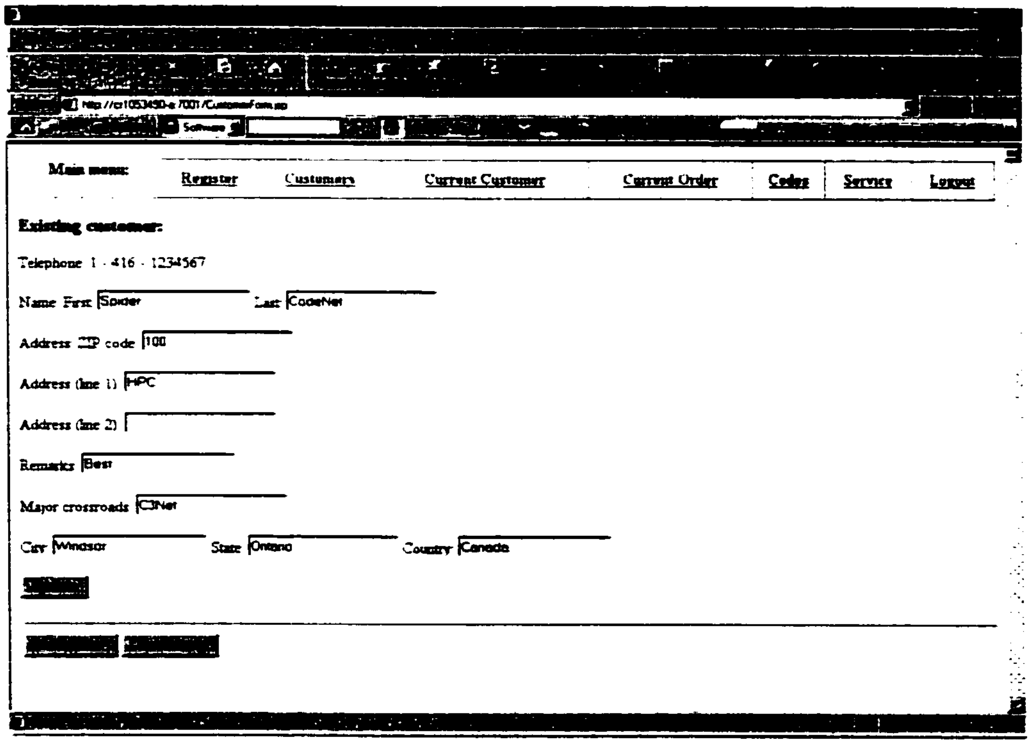


Home

Customers List

First Name	Last Name	ZIP	City	State	Country	Phone
[illegible]	Kent	1	Windsor	ON	Canada	1-519-1234567
[illegible]	Yu	2	Toronto	ON	Canada	1-519-2550197
[illegible]	Zhong	3	Windsor	ON	Canada	1-519-2234567
Paul	Preney	4	Windsor	ON	Canada	1-519-3234567
Nita	Majamuda	5	Windsor	ON	Canada	1-519-4234567
Kameron	Chouidy	6	Windsor	ON	Canada	1-519-5234567
[illegible]	Code	100	Windsor	Ontario	Canada	1-416-1234567

Figure 5.11 Customers List



Existing customer:

Telephone 1 - 416 - 1234567

Name First Spider Last CodeNet

Address ZIP code 100

Address (line 1) HPC

Address (line 2)

Remarks Best

Major crossroads C3Net

City Windsor State Ontario Country Canada

Figure 5.12 User Information Form

Figure 5.13 is logout form. When we need to log on as different user. The server will start another session.

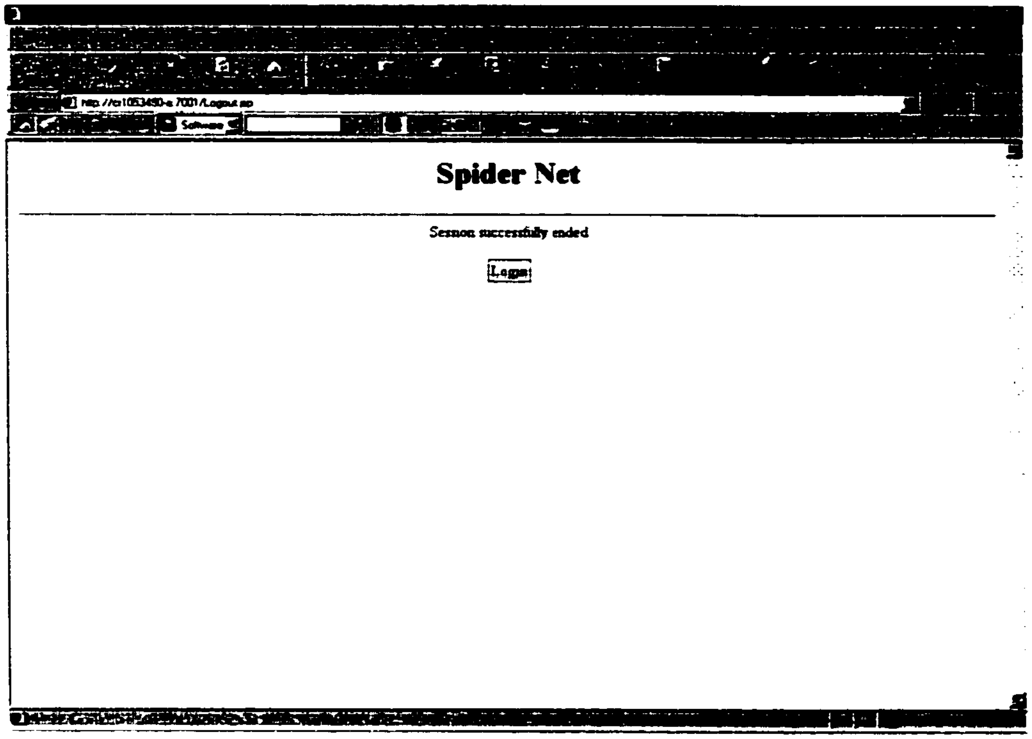


Figure 5.13 Logout Form

Figure 5.14 ask user to input phone number for creating new user record.

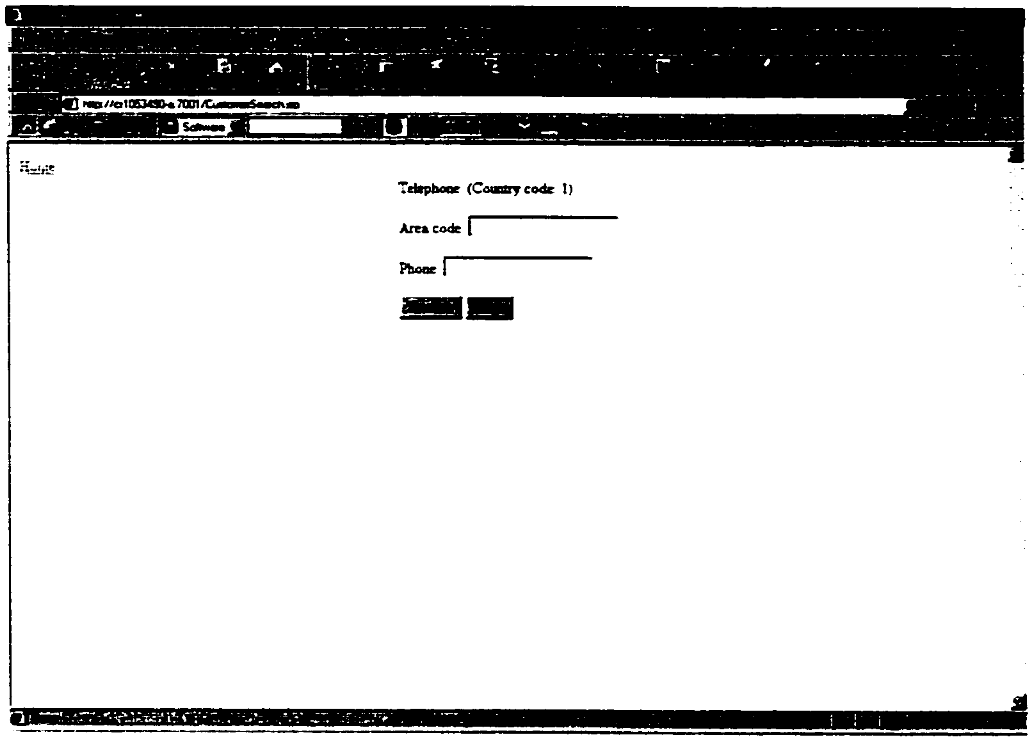
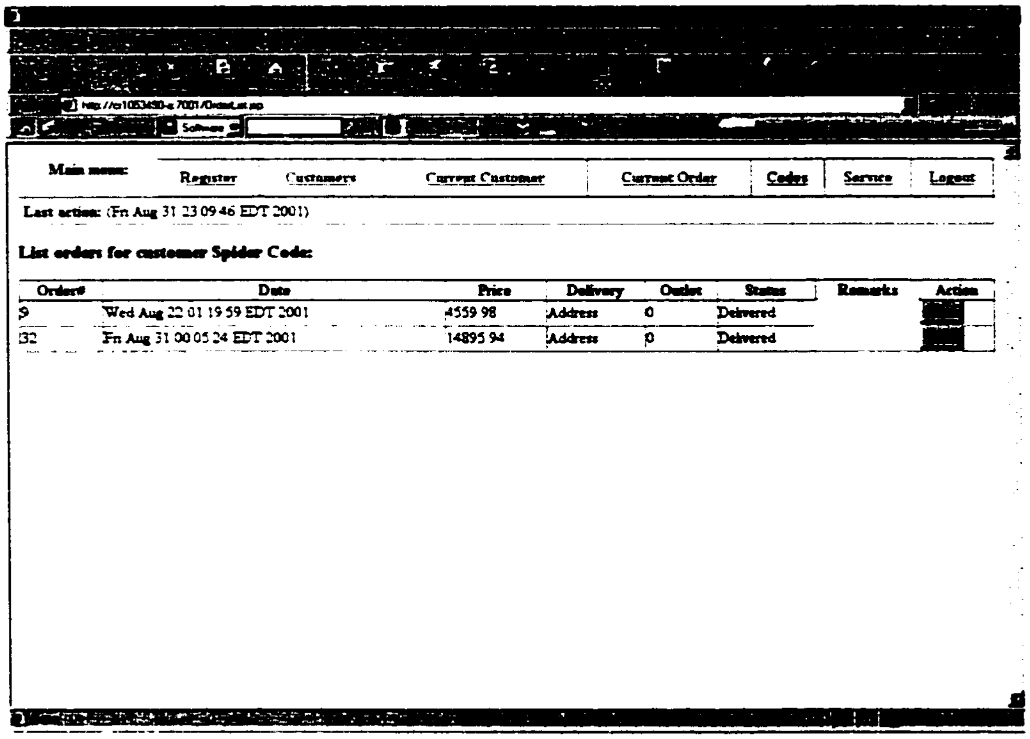


Figure 5.14 Login by Phone Number

When user clicks List Order in Figure 5.12, the system will show all the orders the user has made. It also shows the user the code names and select option box for new order. The prices are shown beside. If the order is not delivered, the user can modify the order information. Figure 5.15 shows this.

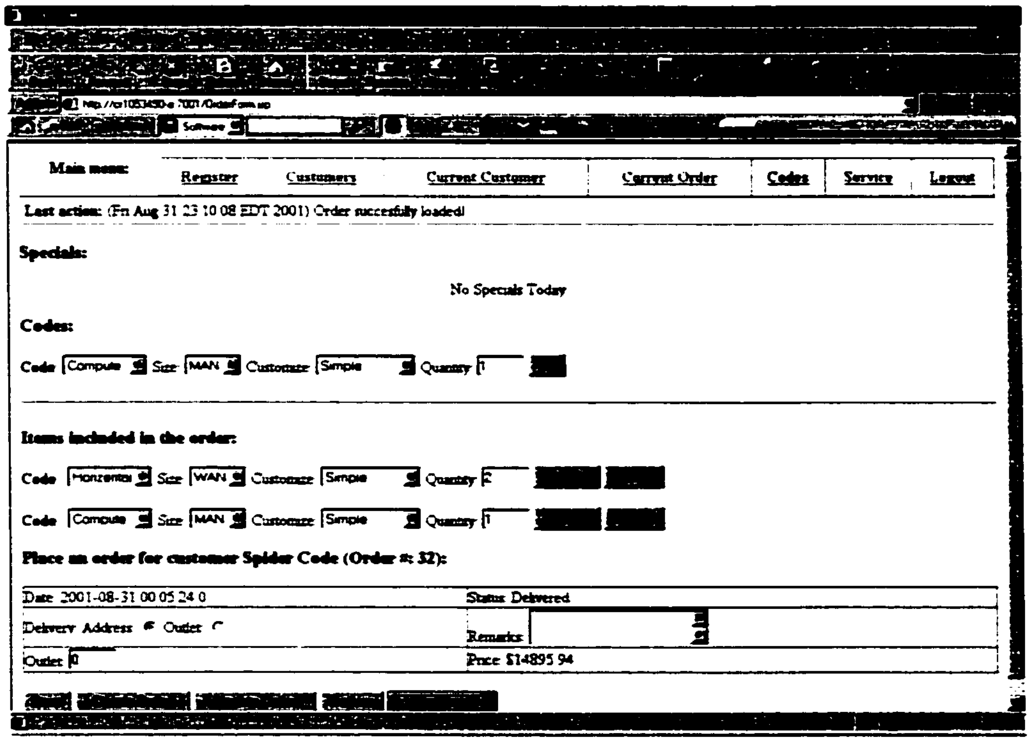


Last action: (Fri Aug 31 23 09 46 EDT 2001)

List orders for customer Spider Code:

Order#	Date	Price	Delivery	Outlet	Status	Remarks	Action
9	Wed Aug 22 01 19 59 EDT 2001	4559 98	Address	0	Delivered		
32	Fri Aug 31 00 05 24 EDT 2001	14895 94	Address	0	Delivered		

Figure 5.15 Order List



Last action: (Fri Aug 31 23 10 08 EDT 2001) Order succesfully loaded!

Specials:

No Specials Today

Codes:

Code Compute Size MAN Customize Simple Quantity 1

Items included in the order:

Code Horizontal Size WAN Customize Simple Quantity 2

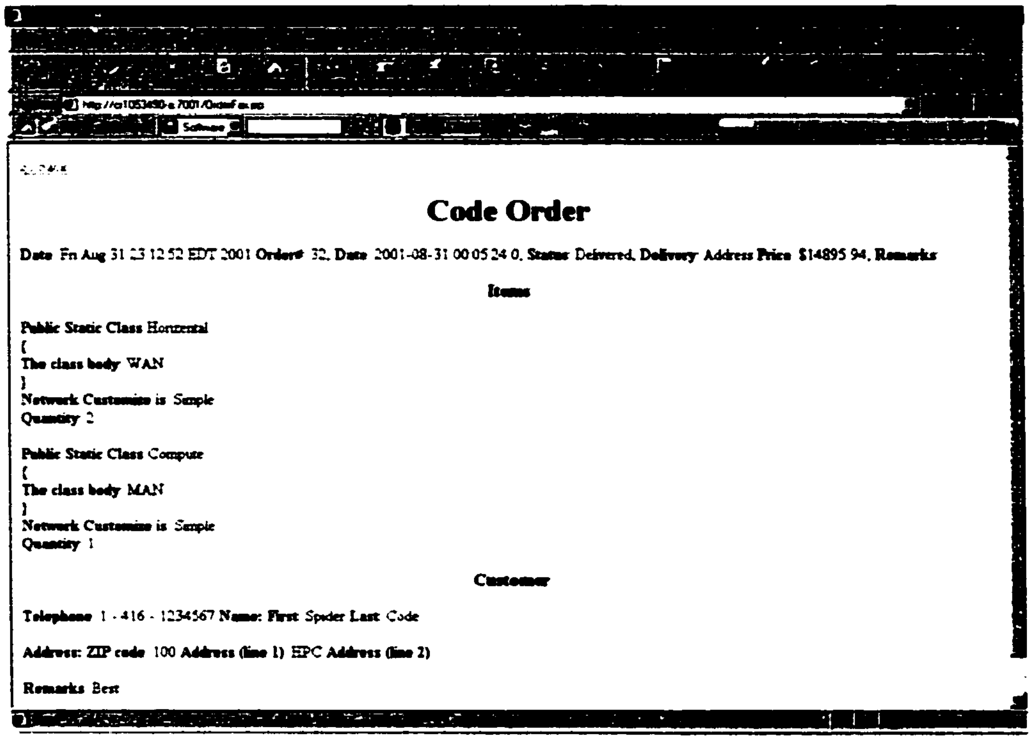
Code Compute Size MAN Customize Simple Quantity 1

Place an order for customer Spider Code (Order #: 32):

Date 2001-08-31 00 05 24 0	Status Delivered
Delivery Address [✓] Outlet	Remarks
Outlet 0	Price $14895 94

Figure 5.16 Order Detail List

Click the "view source". the customer can see the code it generate. Figure 5.17 shows this.



Code Order

Date Fri Aug 31 23 12 52 EDT 2001 **Order#** 32, **Date** 2001-08-31 00 05 24 0, **Status** Delivered, **Delivery Address Price** $14895 94, **Remarks**

Items

Public Static Class Horizontal
{
The class body WAN
}
Network Customize is Simple
Quantity 2

Public Static Class Compute
{
The class body MAN
}
Network Customize is Simple
Quantity 1

Customer

Telephone 1 - 416 - 1234567 **Name: First** Spider **Last** Code

Address: ZIP code 100 **Address (line 1)** EPC **Address (line 2)**

Remarks Best

Figure 5.17 Description of Source Code

Summary of Operations:

A typical use of the system can be the following steps:

(1) A customer connects to the system by pointing the browser to SpiderNet homepage (for example www.spidernet.com)
(2) Figure 5.10 will be shown to the user
(3) Customer browses through the code list. The prices are shown beside. Customer can view the introduction of the code.
(4) Customer adds the chosen code to the order
(5) Customer can see the summary of order including date, customer name, items, total price.
(6) Customer delivers the code. The code is downloaded from the system.

5.10 Limitations

The major limitations of this model are:

- The system depends on WASs. Although different vendors of WAS provide similar functions. They are not always compatible and need different configurations.
- The system has not been tested in multi-server mode due to a license restriction problem. The performance only can be measured in single server with multiple client concurrent connection mode.

We have built the system prototype model. Although it is not complete and many function need to be further developed, it is efficiently verify the theory we have discussed in the previous chapters. In the next chapter, we will give the conclution and future work.

Chapter 6 Conclusions and Future Work

6.1 Contributions and Conclutions

In this thesis, we present a multi-server design model using EJB technology on J2EE platform. This model touches the fundamental problems of grid computation: workload distribution and fault tolerance. The key innovation of this design is trying to use EJB technology for multi-server design.

The following is the summary of the contributions of this thesis:

(1) Give the analysis and design in using EJB for multi-server design.

(2) This design model tries to solve challenging problem in high performance and high availability in multi-server model design.

(3) EJB was used to improve system performance.

(4) Give a development approach based on component and application framework technology. Using WAS as a middle tier, we can use many services from the server instead of building everything from scratch.

(5) The system architecture is flexible. There is much room for future development.

(6) A multi-server prototype model was built.

(7) Thin client solution: the clients only need a Web browser and Java virtual machine in order to use the system service.

(8) JDBC connection pool was used for improve system and database efficiency and performance.

By using WAS as a middle tier and EJB technology for building multi-server, Application system always gets benefit from the development of underlying software framework. There is considerably more work to be done in the area of application framework, specification and EJB technology. The effort on this work is not only benefit for individual application, but also many other application developers and the whole e-business.

6.2 Discussions and Future Development

Things change very fast in computer world. A few years ago, there was no Java or EJB, components were just starting to appear; currently these approaches are becoming mainstream. Based on this prototype model, many further works can be developed. We suggest the following future research directions that need to be further discussed:

- **Integrating EJB and Jini**

  Integrating EJB and Jini for Component-Based Development of Web-based Enterprise Solutions. Combine advantage of these two technologies for E++ solutions.

- **Web Services**

  Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web Services perform the functions that can be anything from simple requests to complicated business processes. A web services platform will allow components/applications built using a variety of application platforms to be integrated in to a single enterprise application. It is an integration technology. Once a Web service is deployed, other applications (and other Web Services) can discover and invoke the deployed service.

  The Web Services approach allows this information to be stored and universally retrieved by any application which requires the information, in a common and well-defined manner, independent of platform and programming environment.

- **Network Modeling Protocol Design for Smart Stub Policy**

  More complicated protocols need to be developed for smart stub policy. Of course, there will be more overhead when one adopts more complicated protocols. But the disadvantages will be overcome by effectively using the server resource and new computer hardware technology.

- **Using Object Oriented Database Instead of Using EJBs**

  Database provides most of the services like transactions, security, concurrency etc. just like any App server. Storing object themselves in the database, with associated business methods, would be far more efficient solution instead of using relational database and then using middleware for objects. Oracle9i AS is an example for this issue.

- **JXTA technology**

  JXTA technology is a set of open, generalized peer-to-peer protocols that allow any connected device on the network from cell phone to PDA from PC to server to communicate and collaborate in a peer to peer manner. With the explosion of content on the network, and the existence of millions of connected devices, a multi-dimensional web or Expanded Web has emerged. Content is both on the edge of the network and in the "deep web". JXTA technology enables new and innovative network applications to be created, giving complete access to content on the Expanded Web.

- **MultiPools**

  JDBC MultiPools create a list of connection pools to be used by a single instance of WAS Server. A configurable algorithm determines which connection is returned. MultiPools provide support for load balancing and high availability. MultiPools make it easier for an application to switch to another RDBMS for distributed processing or during a failover situation.

- **Improving the Efficiency of Java Platform**

  Java has a number of drawbacks in its current form. The most significant is performance. Both EJB and Jini have this problem. How to balance just-in-time and ahead-of-time for optimum performance issue needs to be further researched.

Bibliography

[Amir98] Yair Amir. Baruch Awerbuch. and Ryan S. Borgstrom
'The Java Market: Transforming the Internet into a Metacomputer", 1998

[Anero99] Nikolaos Anerousis
An Architecture for Building Scalable. Web-based Management Services (1999)
http://citeseer.nj.nec.com/

[Arcst01]http://www.io-software.com/products/index.html

[Aron00] Mohit Aron
"Differentiated and Predictable Quality of Service in Web Server Systems", 2000
http://citeseer.nj.nec.com/aron00differentiated.html

[Avers00] Luis Aversa
"Load Balancing a Cluster of Web Servers -- Using Distributed Packet Rewriting". 2000
http://citeseer.nj.nec.com/aversa00load.html

[Booch] Grady Booch
"Object-Oriented Analysis and Design"

[Brad00] Neil Bradley
"The XML Companion". Second Edition. 2000
Addison-Wesley. ISBN:0-201-67486-6

[Bram00] Randall Bramley. Kenneth Chiu.Shridhar Diwan
"A Component Based Services Architecture for Building Distributed Applications"
http://citeseer.nj.nec.com/

[Brown01] Kyle Brown. Dr. Gary Craig, etc
"Enterprise Java Programming with IBM WebSphere"
Addison-Wesley. 2001. ISBN: 0-201-61617-3

[Broy98] Manfred Broy
"A uniform mathematical concept of a component. appendix to M. Broy: What characterizes a software component?"
Software Concepts & Tools. Voleme 19. Number 1. 1998. pp57

[Butle00] Martin Butler
"IBM's Pattern for e-business"
http://www.butlergroup.com/ibm.asp

[Buyya99] Rajkumar Buyya
"High Performance Cluster Computing: Architectures and Systems"

Prentice Hall. ISBN: 0130137847May. 1999

[Buyya01] Rajkumar Buyya and Sudharshan Vazhkudai
"Compute Power Market: Towards a Market-Oriented Grid"
http://citeseer.nj.nec.com/buyya01compute.html

[Calla01] Dustin R. Callaway
"Inside Servlets---Server-Side Programming for the Java Platform, Second Edition"
Addison-Wesley. 2001. ISBN: 0-201-70906-6, pp90

[Carva00] Dulcineia Carvalho. Roy Campbell. Dennis Mickunas
"A Framework in Execution Environments in 2K"
Department of Computer Science. University of Illinois at Urbana-Champaign
http://citeseer.nj.nec.com/445641.html

[CCF00] CCF Project Team. Emory University
"CCF: A Framework for collaborative Computing"
IEEE Internet Computing. January/February 2000.

[Cepon99] Alex Ceponkus & Faraz Hoodbhoy
"Applied XML: A Toolkit for Programmers"
John Wiley & Sons. 1999. Inc. ISBN: 0-471-34402-8

[Chen97] Zhikai Chen Kurt Maly Piyush Mehrotra Praveen K. Vangala Mohammad Zubair
"Web Based Framework for Distributed Computing" (1997)
http://citeseer.nj.nec.com/

[Colaj98] Michele Colajanni Dip. di Informatica
"Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers"
http://citeseer.nj.nec.com/colajanni98dynamic.html

[Elizab00] M. Elizabeth C. Hull. Peter N. Nicholl. Philip Houston, Niall Rooney
"Towards a visual approach for component-based software development"
Software Concept and Tools. Volume 19, Number 4, pp154

[Fayad99] Mohamed E. Fayad. Douglas C. Schmidt, Ralph E. Johnson
"Building Application Frameworks, Object-Oriented Foundations of Framework Design"
John Wiley & Sons, Inc. 1999. ISBN:0-471-24875-4

[Finga00] Peter Fingar
"Component-Based Frameworks for E-Commence"
Communications of the ACM. October 2000/Vol. 43. No 10, pp 63

[Fleis00] Brett D. Fleisch, Heiko Michael, etc
"Fault Tolerance and Configurability in DSM Coherence Protocols"

IEEE Concurrency, April-June 2000, pp10

[Foste00] Ian Foster
"Building the Grid: An Integrated Services and Toolkit Architecture for Next Generation Networked Applications" http://www.gridforum.org/building_the_grid.htm

[Foster96] Ian Foster, Carl Kesselman and Steven Tuecke
"The Nexus Approach to Integrating Multithreading and Communication"
ACM, Journal of Parallel and Distributed Computing37, 70-82 (1996)

[Fred01] Ashley Friedlein
"Web Project Management---Delivering Successful Commercial Web Sites", pp7
Morgan Kaufmann Publishers. 2001, ISBN: 1-55860-678-5

[Gartn00] Felix C.Gartner
"Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments"
ACM Computing Surveys, March 1999, Volume 31

[Globu] "Globus: A Meta Computing Infrastructure Toolkit"
http://citeseer.nj.nec.com/cache/papers2/cs/4154/ftp:zSzzSzinfo.mcs.anl.govzSzpubzSztech_reportszSzreportszSzP614.pdf/foster96globus.pdf

[Goldb98] Adele Goldberg
"A reuse business model"
Software Concepts & Tools, Volume 19, Number 1, 1998, pp11

[Gomez00] Paco Gomez and Peter Zadrozny
"Java 2 Enterprise Edition with BEA Weblogic Server"
Wrox, ISBN: 1-861002-99-8

[Govon99] Darren Govoni
"Java Application Frameworks"
John Wiley & Sons, Inc. ISBN: 0-471-32930-4, pp3,4

[Henri98] Jorg Henrichs
"Optimizing and Load Balancing Metacomputing Applications"
ACM, 1998 International Conference on Supercomputing, July 13-17, 1998, pp165

[IBM] www.redbook.ibm.com
http://www.bm.com/software/developer/

[Ingha00] David B. Ingham and Santosh K. Shrivastava
"Constructing Dependable Web Services"
IEEE Internet Computing, January/February 2000, pp25

[Jini] http://www-unix.mcs.anl.gov/gridforum/jini/

[Kabir00] Mohammed J. Kabir
"Apache Server Administrator's Handbook"
IDG Books ISBN: 0-7645-3306-1 pp15

[Kosti00] Alexander E. Kostin, Member, IEEE, Isik Aybay, and Gurcu Oz
"A Randomized Contention-Based Load-Balancing Protocol for a Distributed Multiserver Queeing System"
IEEE Transactions on Parallel and Distributed Systems, vol.11, no.12, December 2000

[Krish00] Balachander Krishnamurthy, Jia Wang
"On Network-Aware Clustering of Web Clients"
ACM, Computer Communication Review, Sigcomm, Volume 30, Number 4, October 2000, pp97

[Kuzm01] Aleksandar Kuzmanovic and Edward W. Knightly
"Measuring Service in Multi-Class Networks", 2001
http://citeseer.nj.nec.com/kuzmanovic01measuring.html

[Legion] www.legion.virginia.edu

[Lewis] Ted Lewis

"Object-Oriented Application Frameworks"

[Lu00] Paul Lu
"Implelmenting Scoped Behavior for Flexible Distributed Data Sharing"
IEEE, Concurrency, July-September, 2000, pp63

[Manol99] Frank Manola
"Technologies for a Web Object Model"
IEEE Internet Computing , January/February, 1999, pp38

[Maten01] Vlada Matena
"Applying Enterprise JavaBeans ---Component-Based Development for the J2EE Platform"
Addison-Wesley, ISBN:0-201-70267-3

[Monso00] Richard Monson-Haefel
"Enterprise JavaBeans"
O'Reilly, ISBN: 1-56592-869-5, pp 399

[Nilss00] Dale R. Nilsson, Peter M. Jakab, Bill Sarantokos, Russell A. Stinehour
"Enterprise Development with VisualAge for Java, Version 3"
John Wiley & Sons, Inc, ISBN: 0-471-38949-8, pp437

[Peter96] Larry L. Peterson & Bruce S. Davie
"Computer Networks---A System Approach"
Morgan Kaufmann Publishers, Inc. 1996, ISBN:1-55860-368-9

[Roman99] Ed Roman
"Mastering Enterprise JavaBeans and the Java2 Platform, Enterprise Edition"
John Wiley & Sons. ISBN: 0-471-33229-1

[Schne97] Jeff Schnejder
"Using Enterprise Java"
QUE. ISBN: 0-7897-0887-6

[Siege00] Jon Siegel
"CORBA 3 Fundamentals and Programming"
John Wiley and Sons, Inc. ISBN: 0-471-29518-3

[Singh99] Harry Singh
"Progressing to Distributed Multiprocessing"
Prentice Hall PTR. ISBN:0-13-095683

[Solom00] David A. Solomon & Mark E. Russinovich
"Inside Microsoft Windows 2000, Third Edition"
Microsoft Press. 2000. ISBN: 0-7356-1021-5. pp784

[Subra00] Subrahmanyam Allamaraju, Karl Avedal, etc
"Professional Java Server Programming J2EE Edition"
Wrox Press Ltd. 2000. ISBN: 1-861004-65-6

[SUN] Sun Microsystems. http://java.sun.com/products/
http://docs.sun.com/
http://java.sun.com/j2ee/
http://developer.java.sun.com/
Java ™ 2 Platform.Enterprise Edition Blueprints —www.java.sun.com/j2ee/blueprints
Java ™ 2 Platform Enterprise Edition Specification —www.java.sun.com/products
Java ™ Servlet Specificati n,v2.2 —www.java.sun.com/products/servlet/index.html
JavaServer Pages ™ Specification.v1.1 —www.java.sun.com/products/jsp/index.html

[SUN10] http://www.sun.com/products-n-solutions/software/oe-platforms/java2ee.html

[SUN11] http://www.sun.com/us/service/sunps/jdc/java_centers.html

[Talig] Taligent

"Building Object-Oriented Frameworks"

[These01] http://www.theserverside.com
http://www2.theserverside.com/reviews/index.jsp

[Vogel99] Andreas Vogel. Madhavan Rangarao
"Programming with Enterprise JavaBeans, JTS, and OTS:Building Distributed Transactions with Java and C++"
John Wiley & Sons, Inc, ISBN: 0-471-31972-4

[Wahli00] Ueli Wahli. Mitch Fielding, etc
"Servlet and JSP Programming with IBM WebSphere studio and VisualAge for Java"
ibm.com/redbooks. ISBN: 0-7384-1608-8

[BEA] BEA
http://www.bea.com

[Zhang00] Michael Hui Zhang, "Design and Construction of a Distributed Library-Based Software Reuse Model", Thesis Report, 2000

[Zhong00] Andy Sheng Zhong, "Software Library for Reuse-oriented Program Development", Thesis Report, 2000

During the course of thesis. we have occasion to reference a number of websites which contain material related to vary aspects of current development. Although this information was not of a type to be referenced directly, we have included the reference below as useful links:

http://www.itpapers.com/
http://www.jguru.com/faq/ejb
http://www.weblogic.com/doc51/examples/ejb/package-examples.ejb.html
http://safari2.oreilly.com/table.asp?bookname=entjbean2
http://www.javaworld.com/jw-03-1999/jw-03-middleware.html
http://www.javaworld.com/javaworld/jw-04-1999/jw-04-middleware.html
http://citeseer.nj.nec.com/cache/papers2/cs/16505/http:zSzzSzjava.sun.comzSzproductsz
SzejbzSzpdfzSzj2ee_dnatwp.pdf/roman99technical.pdf
http://www.javamagazin.de/
http://www.ohioedge.com/

Appendix A: SpiderNet Source Code Definition

This appendix contains the SpiderNet source code definition. We have the root directory $SpiderCode. The files are organized with directories. Each directory of the source files corresponds to a java package. All the Java source files are stored in directory src and the compiled Java class files are stored in directory srv.

Java Classes in Package xiaohong.yu (\SpiderCode\src\yu\java)

Files: Code. CodeGeneration. CodeServicesSL, CodeServiceSLBean. CodeServicesSLHome. Customer. CustomerListS. CustomerMaintenancesS. CustomerSearchS. Helen. OrderDetail, OrderMaintenceS, OrderMaser, OrderServiceSL. OrderServiceSLBean. OrderServiceSLHome, OrderSF, OrderSFBean, Unit

JSP Files (\SpiderCode\src\yu\jsp)

CodeList. CustomerForm. CustomerSearch. Index. Logout. Message. OrderForm. OrderList. OrderSubForm. Specials

Servlet Files (\SpiderCode\src\yu\servlet)

CustomerListS. CustomerMaintenceS. CustomerSearchS. OrderMaintenceS. ServiceMaintenceS

XML Description Files

The description files are stored in CodeServicesSL, OrderServicesSL. ServicesSL, OrderSF. CodeEB and UnitEB sub directories. Each sub directory contains two files: ejb-jar and weblogic-jar. We already gave the description file of CodeServicesSL EJB, the description files of the other EJBs are listed in the followings:

OrderServicesSL
ejb-jar:

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
   <session>
      <ejb-name>xiaohong.yu.OrderServicesSL</ejb-name>
```

```
        <home>xiaohong.yu.OrderServicesSLHome</home>
        <remote>xiaohong.yu.OrderServicesSL</remote>
        <ejb-class>xiaohong.yu.OrderServicesSLBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
   </session>
  </enterprise-beans>
  <assembly-descriptor>
   <container-transaction>
        <method>
         <ejb-name>xiaohong.yu.OrderServicesSL</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
   </container-transaction>
  </assembly-descriptor>
 </ejb-jar>
```

WebLogic-jar

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
   <ejb-name>xiaohong.yu.OrderServicesSL</ejb-name>
   <caching-descriptor>
        <max-beans-in-free-pool>100</max-beans-in-free-pool>
   </caching-descriptor>
   <jndi-name>xiaohong.yu.OrderServicesSL</jndi-name>
  </weblogic-enterprise-bean>
 </weblogic-ejb-jar>
```

OrderSF

ejb-jar

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
   <session>
        <ejb-name>xiaohong.yu.OrderSF</ejb-name>
        <home>xiaohong.yu.OrderSFHome</home>
        <remote>xiaohong.yu.OrderSF</remote>
        <ejb-class>xiaohong.yu.OrderSFBean</ejb-class>
        <session-type>Stateful</session-type>
        <transaction-type>Container</transaction-type>
   </session>
  </enterprise-beans>
  <assembly-descriptor>
```

```
    <container-transaction>
         <method>
          <ejb-name>xiaohong.yu.OrderSF</ejb-name>
          <method-intf>Remote</method-intf>
          <method-name>*</method-name>
         </method>
         <trans-attribute>Required</trans-attribute>
    </container-transaction>
   </assembly-descriptor>
  </ejb-jar>
```

webLogic-jar

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
    <ejb-name>xiaohong.yu.OrderSF</ejb-name>
    <caching-descriptor>
         <max-beans-in-cache>300</max-beans-in-cache>
         <idle-timeout-seconds>60</idle-timeout-seconds>
    </caching-descriptor>
    <jndi-name>xiaohong.yu.OrderSF</jndi-name>
   </weblogic-enterprise-bean>
  </weblogic-ejb-jar>
```

CodeEB

ejb-jar

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
   <enterprise-beans>
    <entity>
         <ejb-name>xiaohong.yu.CodeEB</ejb-name>
         <home>xiaohong.yu.CodeEBHome</home>
         <remote>xiaohong.yu.CodeEB</remote>
         <ejb-class>xiaohong.yu.CodeEBBean</ejb-class>
         <persistence-type>Bean</persistence-type>
         <prim-key-class>xiaohong.yu.CodeEBPK</prim-key-class>
         <reentrant>False</reentrant>
    </entity>
   </enterprise-beans>
   <assembly-descriptor>
    <container-transaction>
         <method>
          <ejb-name>xiaohong.yu.CodeEB</ejb-name>
          <method-intf>Remote</method-intf>
          <method-name>*</method-name>
         </method>
```

```
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
   </assembly-descriptor>
 </ejb-jar>
```

weblogic-jar

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
    <ejb-name>xiaohong.yu.CodeEB</ejb-name>
    <caching-descriptor>
        <max-beans-in-cache>100</max-beans-in-cache>
        <idle-timeout-seconds>5</idle-timeout-seconds>
    </caching-descriptor>
    <persistence-descriptor>
        <delay-updates-until-end-of-tx>false</delay-updates-until-end-of-tx>
    </persistence-descriptor>
    <jndi-name>xiaohong.yu.CodeEB</jndi-name>
   </weblogic-enterprise-bean>
 </weblogic-ejb-jar>
```

UnitEB

ejb-jar

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
   <enterprise-beans>
    <entity>
        <ejb-name>xiaohong.yu.UnitEB</ejb-name>
        <home>xiaohong.yu.UnitEBHome</home>
        <remote>xiaohong.yu.UnitEB</remote>
        <ejb-class>xiaohong.yu.UnitEBBean</ejb-class>
        <persistence-type>Bean</persistence-type>
        <prim-key-class>xiaohong.yu.UnitEBPK</prim-key-class>
        <reentrant>False</reentrant>
    </entity>
   </enterprise-beans>
   <assembly-descriptor>
    <container-transaction>
        <method>
          <ejb-name>xiaohong.yu.UnitEB</ejb-name>
          <method-intf>Remote</method-intf>
          <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
   </assembly-descriptor>
```

```
 </ejb-jar>
```

weblogic-jar

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
     <ejb-name>xiaohong.yu.UnitEB</ejb-name>
     <caching-descriptor>
          <max-beans-in-cache>100</max-beans-in-cache>
          <idle-timeout-seconds>5</idle-timeout-seconds>
     </caching-descriptor>
     <persistence-descriptor>
          <delay-updates-until-end-of-tx>false</delay-updates-until-end-of-tx>
     </persistence-descriptor>
     <jndi-name>xiaohong.yu.UnitEB</jndi-name>
   </weblogic-enterprise-bean>
 </weblogic-ejb-jar>
```

ServiceSL
ejb-jar

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
   <enterprise-beans>
     <session>
          <ejb-name>xiaohong.yu.ServicesSL</ejb-name>
          <home>xiaohong.yu.ServicesSLHome</home>
          <remote>xiaohong.yu.ServicesSL</remote>
          <ejb-class>xiaohong.yu.ServicesSLBean</ejb-class>
          <session-type>Stateless</session-type>
          <transaction-type>Container</transaction-type>
     </session>
   </enterprise-beans>
   <assembly-descriptor>
     <container-transaction>
          <method>
            <ejb-name>xiaohong.yu.ServicesSL</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>*</method-name>
          </method>
          <trans-attribute>Required</trans-attribute>
     </container-transaction>
   </assembly-descriptor>
 </ejb-jar>
```

weblogic-jar

```
<?xml version="1.0"?>
```

```
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
     <ejb-name>xiaohong.yu.ServicesSL</ejb-name>
     <caching-descriptor>
          <max-beans-in-free-pool>100</max-beans-in-free-pool>
     </caching-descriptor>
     <jndi-name>xiaohong.yu.ServicesSL</jndi-name>
   </weblogic-enterprise-bean>
  </weblogic-ejb-jar>
```

VITA AUCTORIS

Xiaohong Yu was born in 1966 in Fuqing, P.R. China. He graduated from Fuqing No.1 High School in 1983. From there he went on to Fuzhou University where he obtained a B.C.S . in Computer Science in 1987. He is currently a candidate for the Master's degree in Computer Science at University of Windsor and will graduate in the fall of 2001.