ENABLING TECHNOLOGIES FOR A WEB-BASED URBAN STREET CONSTRUCTION PERMIT SYSTEM

BY
CHANGXIN QI

B.S., CIVIL ENGINEERING
ZHEJIANG UNIVERSITY (1986)

M.S., CIVIL ENGINEERING
ZHEJIANG UNIVERSITY (1991)

SUBMITTED TO THE DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING
In Civil and Environmental Engineering

AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUNE 2001

Signature of Author ______________________________
Department of Civil and Environmental Engineering
May 11, 2001

Certified by ______________________________
George Kocur
Senior Lecturer in Civil and Environmental Engineering
Thesis Supervisor

Accepted by ______________________________
Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies

ENABLING TECHNOLOGIES FOR A WEB-BASED URBAN STREET CONSTRUCTION PERMIT SYSTEM

By
Changxin Qi

Submitted to the Department of Civil and Environmental Engineering on May 11, 2001,
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering
in Civil and Environmental Engineering

ABSTRACT

This thesis is focused on the enabling technologies for a web-based urban street construction permit system. The web-based application system can automatically verify the various constraints, issue the permit if the constraints are met, notify the relevant persons of the issuance of the permit, update the pavement status for the affected street and prepare the billing report for further processing with the existing billing system.

The web-based permit system is divided into two sub-systems: External System and Internal System. The external system is used by contractor/utility companies for permit application, and the internal system is used solely by authorized internal users for maintenance of the system or permit application on behalf of contractor/utility companies when there is such a necessity. These two sub-systems share the same underlying database system.

In order to develop this web-based permit system, the following J2EE technologies have been used: Enterprise JavaBeans, JavaServer Pages, Servlet and JDBC API. Other J2EE technologies such as Transaction, JNDI and XML are also discussed where appropriate.

The following development environments to support these technologies are also presented in this thesis: Red Hat Linux 7.0, Java 2 Platform, Tomcat Server 3.2.1, Database MySQL 2.1.4, and JDBC Driver 2.0.4 for MySQL.

As an example, Arlington permit system was used to demonstrate the design of an Entity-Relationship model, and an Enterprise JavaBeans application.

Thesis Supervisor: George Kocur, Ph.D.
Title: Senior Lecturer in Civil and Environmental Engineering

Acknowledgments

I would like to thank my thesis advisor, Prof. George Kocur, for his support and advising throughout the development of my thesis. George has an amazing ability in working efficiently with deep insight into the subject, which inspired me a lot. It has been a wonderful experience to do research under his guidance.

My thanks also to Andreas Klimke and Rajesh Prasad, my friends who worked with me on the Arlington Permit System Project, and Ron Santosuosso, the director of the Arlington Public Works Department Engineering Division who provided us help for the Arlington Project.

Special thanks go to my wife Jian, my six-years-old daughter Han-Xiao, and my four-months-old son Sheng-Kai for their love, support and smile.

Table of Contents

Table of Figures

List of Tables

1 Introduction

1.1 Overview of the Web-based Permit System

Any person who needs to perform road construction on the public roads of a town or city generally must first obtain a permit from the Public Works Department of the city or town. The most frequent type of road construction work is opening an excavation, as required by utility companies (telco, gas, cable TV). The work is performed either by the utility company itself, or by a contractor. Usually the permit issuing procedure is tedious and inefficient, because the person(s) or firm performing the work or their representative have to physically go to the Public Works Department for the permit, and at least one staff member always needs to be in the Public Works Department office to issue a permit. Also, the validating of the street and dates for the permit has to be done manually, since there are restrictions on certain streets and dates during which no construction may be allowed.

Recent years have witnessed the rapid development of computer technologies, and the Internet growth is also astonishing. According to the graph shown in Figure 1-1 [Zakon2001], the number of web sites increased 200 times from early 1996 to early 2001! As the number of people within the United States connected to the Internet increases steadily, more and more municipal governments have utilized a Web site as one of their main information vehicles to the public.

As a public access initiative, the town of Arlington, Massachusetts provides a web site [Arlington2001a] (http://www.town.arlington.ma.us/) for official and unofficial information. The web site provides on-line town bill lookup as well as information about weather, schools, the Fire Department, Treasurer's Office and many other town functions.

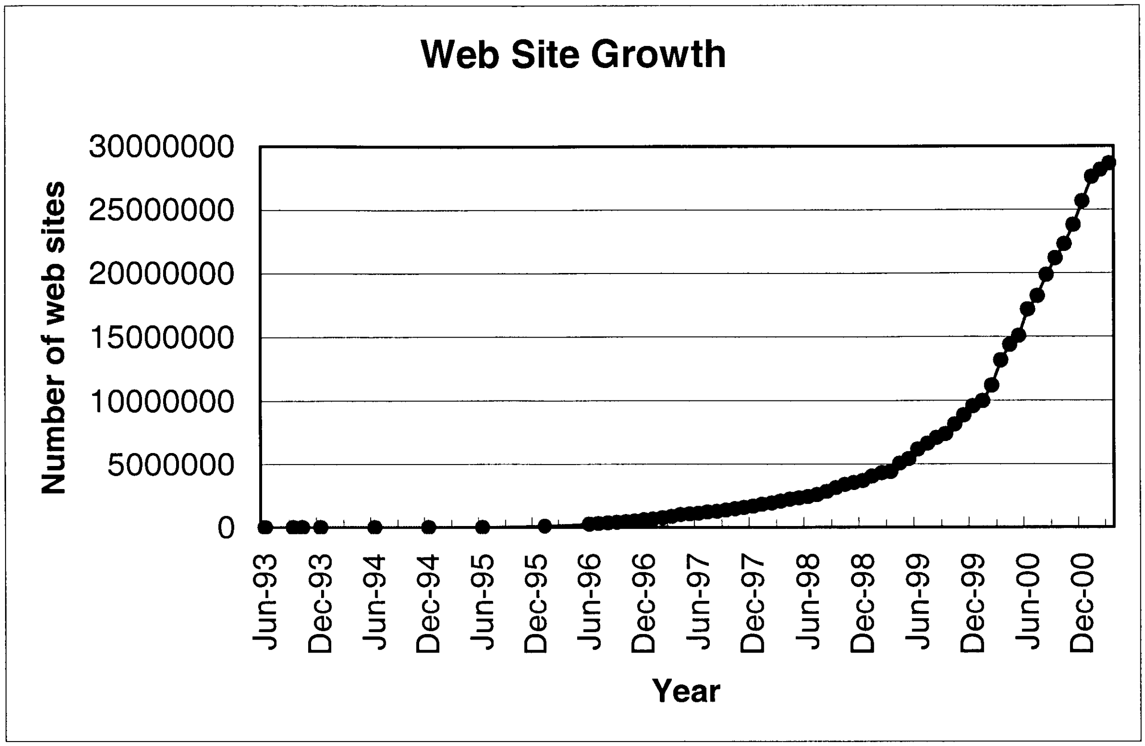
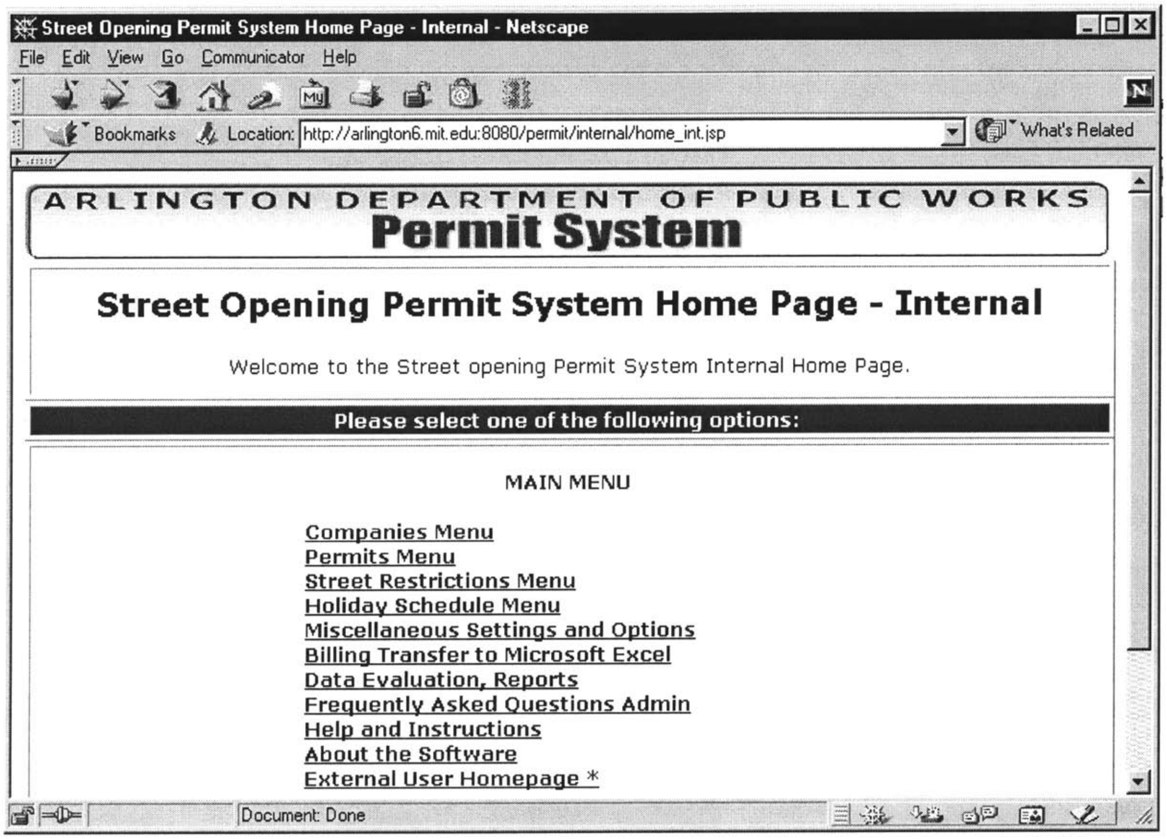


Figure 1-1 Web Site Growth

A natural extension of the above services is a web-based on-line street opening permit issuing system. The web-based permit application form will be automatically evaluated to meet constraints set by the Arlington Department of Public Works. This includes verification of streets and addresses, verification of the proposed dates for the street opening etc. The permit will then automatically be issued on-line. Furthermore, the system will update the pavement status for the affected street (e.g., the Arlington pavement management system will be notified of any additional patches, manholes or gates added to the street). Monthly and weekly reports will be prepared by the system upon request. Billing information will be generated for further processing with the existing billing system.



Street Opening Permit System Home Page - Internal - Netscape
File Edit View Go Communicator Help
Bookmarks
Location:
What's Related
Document: Done

http://arlington6.mit.edu:8080/permit/internal/home_int.jsp

ARLINGTON DEPARTMENT OF PUBLIC WORKS
Permit System

Street Opening Permit System Home Page - Internal

Welcome to the Street opening Permit System Internal Home Page.

Please select one of the following options:

MAIN MENU

Companies Menu
Permits Menu
Street Restrictions Menu
Holiday Schedule Menu
Miscellaneous Settings and Options
Billing Transfer to Microsoft Excel
Data Evaluation, Reports
Frequently Asked Questions Admin
Help and Instructions
About the Software
External User Homepage *

Figure 1-2 Main Menu of Internal Permit System

Figure 1-2 and Figure 1-3 show some screen shots of the Arlington web-based street opening permit system [Arlington2001b]. The advantages of the permit system are:

(1) Allowing contractors and utilities to obtain permits via the Web instead of coming to the DPW office, or receiving permits via fax.

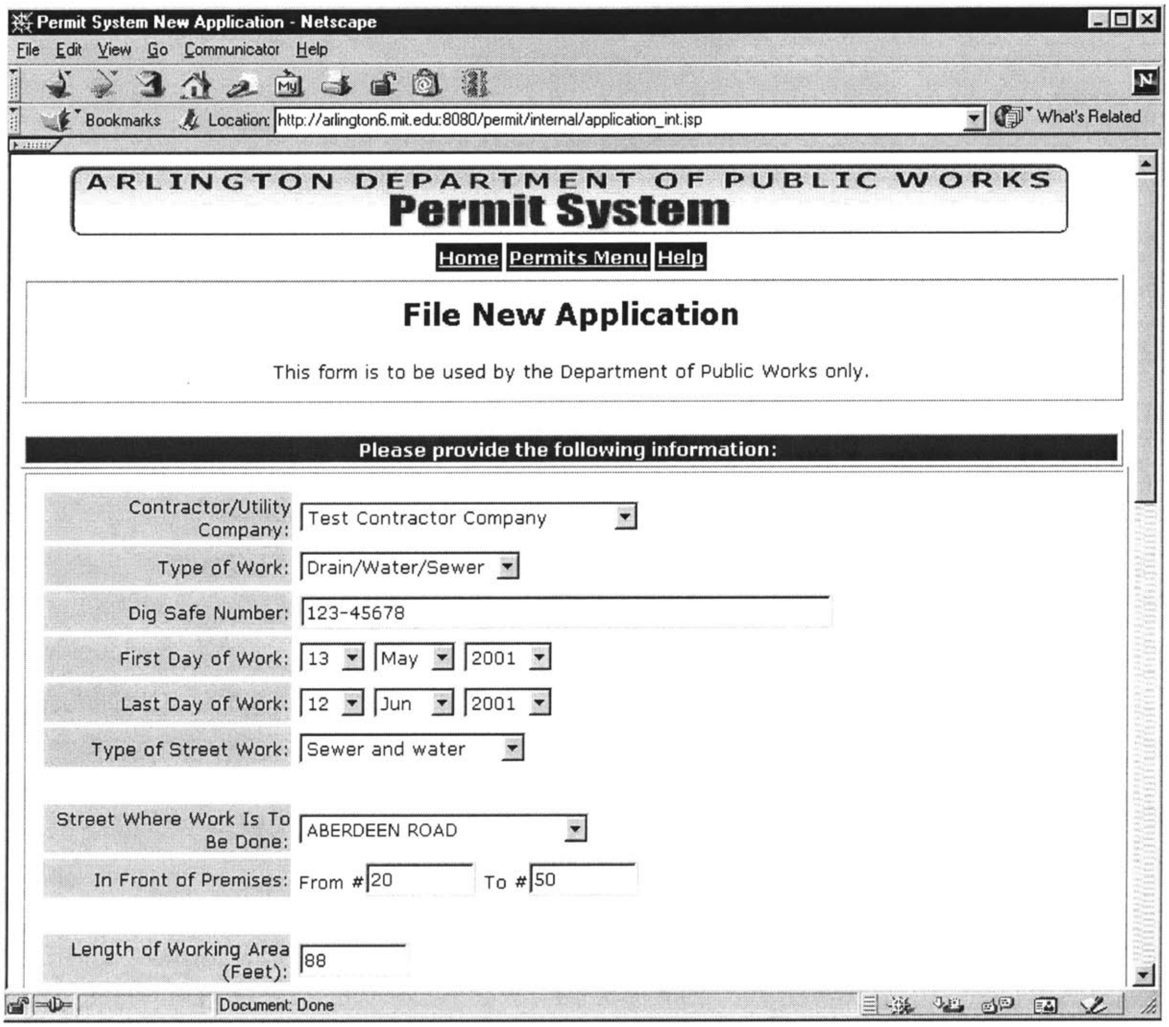
(2) Eliminating the need to have a DPW staff member always available at the counter to process applications.

(3) Automating the procedure of validating the street address and the date constraints.

(4) Automatically updating the relevant data in the pavement management system such as the number of manholes and gates for the convenience of maintenance and statistics purpose.

(5) Automatic report generation periodically

(6) Automatic billing information transfer


Permit System New Application - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://arlington6.mit.edu:8080/permit/internal/application_int.jsp
What's Related
Home
Permits Menu
Help
Document: Done

ARLINGTON DEPARTMENT OF PUBLIC WORKS
Permit System

File New Application

This form is to be used by the Department of Public Works only.

Please provide the following information:

Contractor/Utility Company:	Test Contractor Company
Type of Work:	Drain/Water/Sewer
Dig Safe Number:	123-45678
First Day of Work:	13 May 2001
Last Day of Work:	12 Jun 2001
Type of Street Work:	Sewer and water
Street Where Work Is To Be Done:	ABERDEEN ROAD
In Front of Premises:	From # 20 To # 50
Length of Working Area (Feet):	88

Figure 1-3 Permit Application Page

1.2 Multi-tier Software Architecture

The three-tier software architecture emerged in the 1990s to overcome the limitations of the two-tier architecture. The middle tier server is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users by providing functions such as queuing, application execution, and database staging. The three-tier architecture is used when an effective distributed client/server design is needed that provides increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. These characteristics have made three tier architectures a popular choice for Internet applications and net-centric information systems.

Some Internet applications have light clients written in HTML and application servers written in C++ or Java, and the gap between these two layers is too big to link them together. Instead, there is an intermediate layer (web server) implemented in a scripting language. This layer receives requests from the Internet clients and generates html using the services provided by the business layer. This additional layer provides further isolation between the application layout and the application logic, so sometimes people call it multi tier architecture instead of three-tier architecture to show that there are additional layer(s).

A multi-tier architecture facilitates software development because each tier can be built and executed on a separate platform, thus making it easier to organize the implementation. Also, the architecture makes it easier for different tiers to be developed in different languages, such as a graphical user interface language or light internet clients (HTML, applets) for the top tier; C/C++, Java or SmallTalk for the middle tier; and SQL for much of the database tier.

The solution provided by Sun Microsystems, Inc is the J2EE architecture (The Java 2 SDK, Enterprise Edition). Fig. 1-4 shows the J2EE application model [SUN1999]:

In our development of web-based street opening permit system, we use enabling technologies such as JSP, Java servlets, and JDBC to create our application model. We did not use Enterprise Java Beans given the limited scale of our application, but this thesis explores their usage in applications for scalability, transparency and other issues.

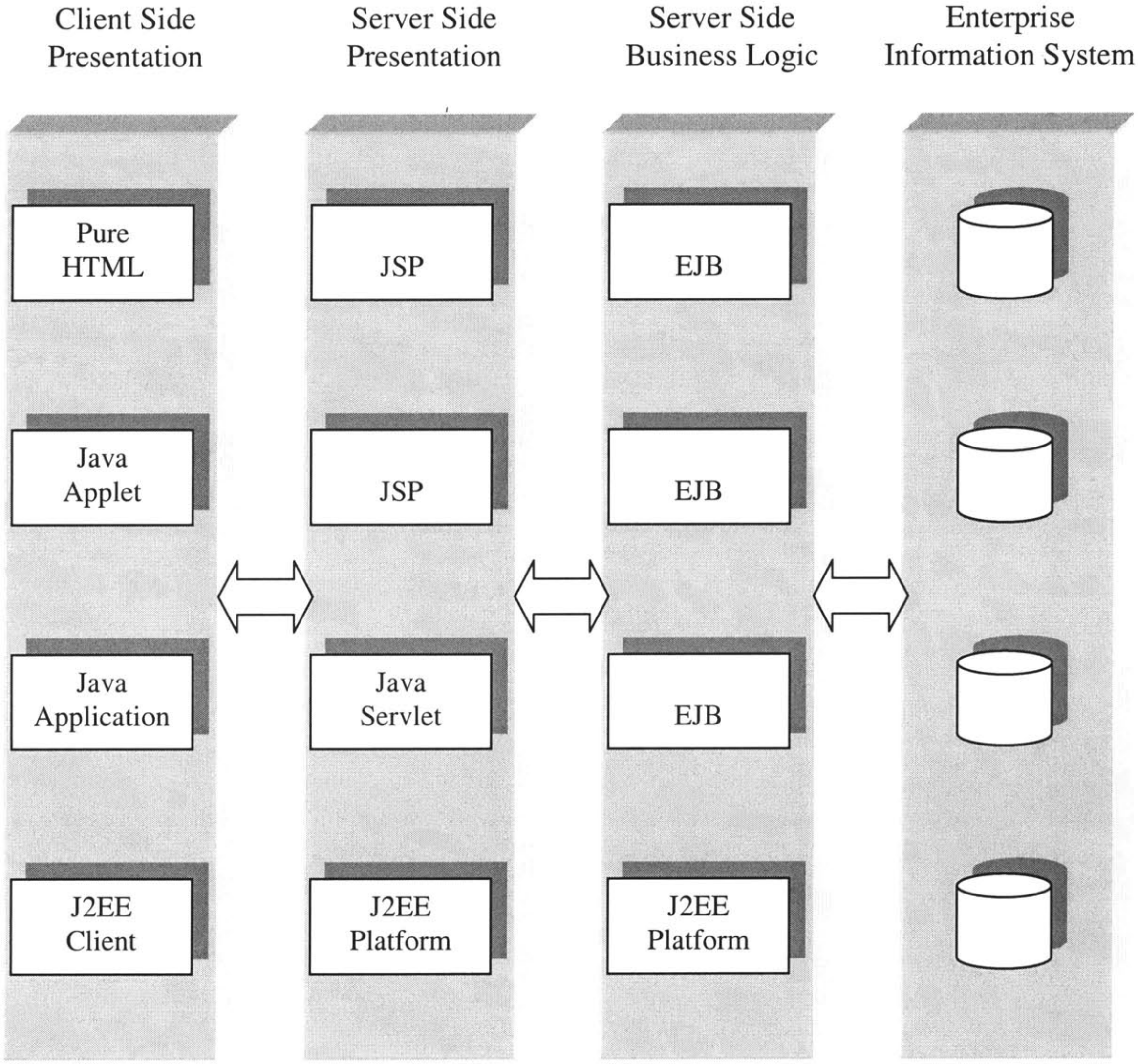


Figure 1-4 J2EE Application Model

1.3 Outline of This Thesis

This first chapter has described the existing problems in current street opening permit issuing practice and introduced our solution to the problem by developing an on-line permit issuing system. We also briefly introduced the multi tier architecture that will be used as the infrastructure of our web-based permit system. The remaining part of the thesis is organized as follows:

Chapter 2 focuses the development environment. It describes the installation of the operating system, Java software development kit, web server, database management system and its driver. It provides guidance as well as a record of the actual installation procedure, and thus is helpful for future deployment in similar environments.

Chapter 3 discusses the J2EE platform technologies---JDBC API, web components and EJB components. By using the JDBC API, we can access a wide variety of different SQL databases with the same Java syntax. Web components, servlet and JSP technology will also be discussed briefly in this chapter. The EJB architecture and enterprise application model are described. The advantages of EJB technology as well as the EJB developer roles will also be discussed.

Chapter 4 deals with the design of the database that provides the core of the whole system. Using the Arlington permit system as an example, the Entity-Relationship model is presented; normalization and denormalization are also discussed.

As a case study of the Arlington permit system, Chapter 5 demonstrates how the Enterprise JavaBeans Technology can be used in the online street construction permit system. A ProcessPermit session bean and a Permit entity bean as well as web client are used for the analysis, and a step-by-step procedure of deploying EJB application are also illustrated.

Chapter 6 gives a short summary of the work and possible future work in this area.

2 Development Environment

With our client (Arlington Department of Public Works) in mind, we decided to develop our web-based street permit system using open source tools. We use the Linux operating system, Java 2 platform, Tomcat server, MySQL database management system and corresponding JDBC drivers. In this chapter, each of these components is introduced and some of the installation procedures are also briefly described.

2.1 Red Hat Linux 7.0

Linux is a free, open source UNIX like operating system designed for personal computers. Linux is not UNIX, but it does share UNIX's command set and look-and-feel, so if you know UNIX, then you know Linux, and vice versa.

Linus Torvalds at the University of Helsinki in Finland originally developed Linux, but now most of Linux is from the Free Software Foundation so numerous programmers worldwide helped develop the operating system.

Nearly 60% of Linux installations either are Red Hat or based on Red Hat Linux. One of the main features of the installation package is the Red Hat Packet Manager (RPM). Using the RPM enables you to safely install and uninstall packages. Another main feature is the ability to install from an X session.

The operating system we used is Red Hat Linux 7[RedHat200]. It has the following new features and enhancements:

2.1.1 Installation-Related Enhancements

(1) The Red Hat Linux 7 installation program is capable of installing Red Hat Linux from multiple CD-ROMs.

(2) The Red Hat Linux 7 installation program supports the newest version of the RPM Package Manager: RPM 4.0.

(3) Xconfigurator helps to configure the X Window System during the installation more thoroughly than before.

(4) The installation program now supports USB devices (mice and keyboards).

(5) Help text has been added to the text mode installation program in the new version.

(6) LDAP and Kerberos can now be configured for account authentication at install-time.

2.1.2 System-Related Enhancements

(1) Many core system components have been upgraded in Red Hat Linux 7.

(2) Many drivers are added or updated to Red Hat Linux 7.

(3) The newest version of the RPM Package Manager is included in Red Hat Linux 7. RPM 4.0 now includes a virtualized database access framework.

(4) All packages that comprise Red Hat Linux 7 have been optimized for maximum performance.

(5) Red Hat Linux 7 includes support for USB. The usbview graphical tool is also included and can be used to display the devices present on the Universal Serial Bus.

(6) Expanded hardware accelerated 3-D support is included in Red Hat Linux 7.

(7) Red Hat Linux 7 has many other new features related to encryption, communication and window manager etc, etc.

2.1.3 Red Hat Linux 7 Installation procedure

The installation of Red Hat Linux 7 is straightforward. Here we briefly describe the major steps in the installation procedure:

(1) Determining Language, Keyboard, and Mouse Settings

The first several screens during the installation are used to gather important information about the input such as the language, keyboard, and mouse device we want to use.

(2) Choosing Installation Class

The install type screen that follows gives the user the option to select an installation class. These options are as follows:

> *Workstation:* This option will automatically install Linux with X and the KDE and/or GNOME desktop environments.
>
> *Server System:* This path will automatically install Linux with additional server packages, including the Apache Web server. X is not automatically installed in this option.
>
> *Custom System.* Custom enables the user to select exactly which packages to install.
>
> *Upgrade*: Upgrade will let you keep your current Linux installation on your hard drive and will simply upgrade your current version of Red Hat Linux to 7.

In setting up the Arlington server, we chose the Custom System because we need to customize some options as we can see in the following description.

(3) Selecting Partition Options

In this step, you need to decide whether to partition your hard drive by yourself or have Linux do it. If you want to manually partition the Linux portion of the hard drive yourself, you have the choice of two tools: Disk Druid and Linux fdisk.

In setting up the Arlington server, we choose to manually partition with the Disk Druid option.

Choose "add" to add the Linux partitions. Create the partitions according to the following recommendations:

> *A swap partition* (16MB, minimum) -- Swap partitions are used to support virtual memory. In other words, data is written to a swap partition when there is not enough RAM to store the data your system is processing. The minimum size of your swap partition should be equal to your computer's RAM, or 16MB (whichever is larger).
>
> *A /boot partition* (16MB, maximum) -- The partition mounted on /boot contains the operating system kernel which allows your system to boot Red Hat Linux, along with files used during the bootstrap process. This partition should be no larger than 16MB.
>
> *A variable-sized root partition* (You can assign the rest of your available disk space to the root partition) -- This is where "/" (the root directory) resides. If only the above three partitions are created, all files except those stored in /boot will reside on the root partition.

(4) Networking Your Computer

In this step of the installation procedure, you need to provide the networking configuration information about your computer such as IP address, netmask, gateway, and DNS etc.

(5) Setting the Time Zone

(6) Configuring Your Accounts

Create the root account and any additional user account in this step. You need to prepare the root password and passwords for any additional accounts.

(7) Selecting Package Groups

In this step, you decide what packages will be installed on your Linux machine. In setting up Arlington server, we checked the option of "everything".

(8) Configuring the X Window System

In this step, you need to configure the X window system to determine how X coordinates with your graphics card and monitor to generate graphic displays.

There are several sub-steps:

a. Choose monitor & sync rates - default settings usually work

b. Choose graphic card, amount of video memory, then test.

c. Choose graphical logon and choose Gnome or KDE as you wish.

(9) Finishing the Installation

Linux will format, then install on disk and then finish the installation.

2.2 Java 2 Platform

The Java 2 Platform provides a comprehensive, end-to-end architecture for building and deploying network-centric applications for the customer. Currently there are three editions for Java 2 Platform, i.e., Standard Edition, Enterprise Edition and Micro Edition. We focus on the Standard Edition and Enterprise Edition.

2.2.1 Java 2 Platform, Standard Edition 1.3 (J2SE)

J2SE provides an infrastructure for building and deploying network-centric enterprise applications for various computers, from the PC to the workgroup server[SUN2000a]. The J2SE is implemented by the Java 2 Software Development Kit (SDK), Standard Edition and the Java 2 Runtime Environment, Standard Edition.

The J2SE provides a stable, secure and feature-complete development and deployment environment designed for the Web. It provides cross-platform compatibility, safe network delivery, and scalability. It facilitates rapid application development. Besides, J2SE version 1.3 improves performance greatly. Figure 2.1 shows the components of Java 2 SDK, Standard Edition v.1.3.

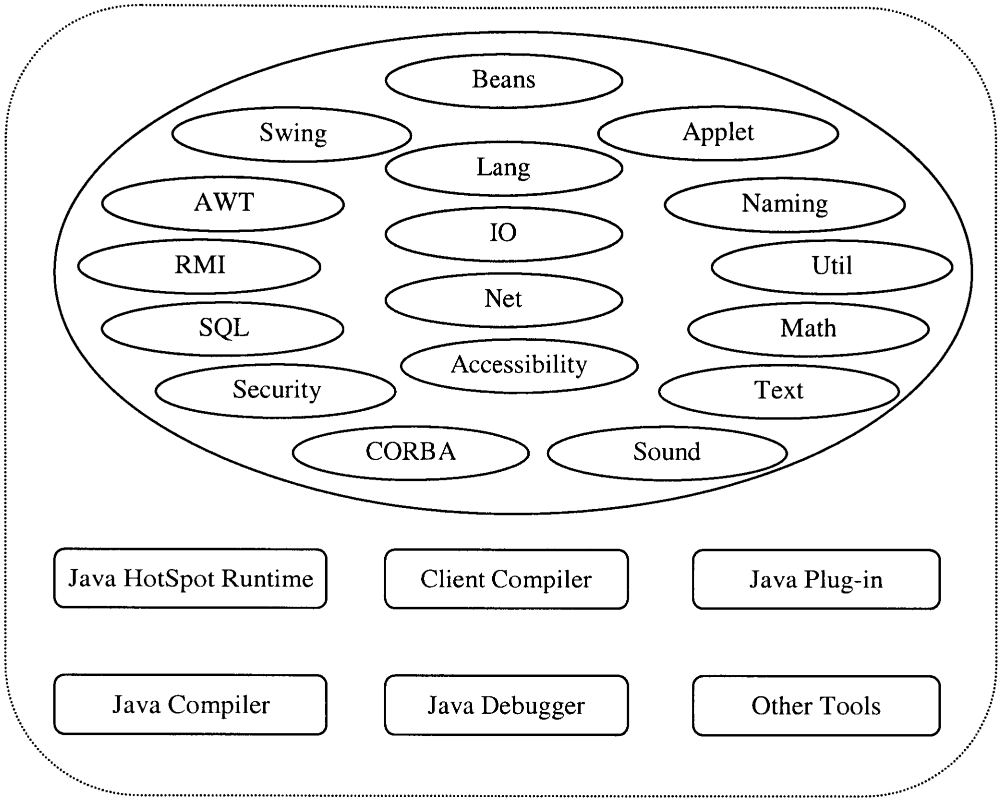


Figure 2-1 Components of Java 2 SDK, Standard Edition v. 1.3

Installation procedure:

1. Go to http://java.sun.com/j2se/1.3/download-linux.html [SUN2000b] and choose RedHat RPM shell script for downloading.
2. Launch the executable file downloaded, j2sdk-1_3_0-linux-rpm.bin, by using the following commands from the directory in which it is located:

   ```
   chmod a+x j2sdk-1_3_0-linux-rpm.bin
   ./j2sdk-1_3_0-linux-rpm.bin
   ```

   The script will create the file j2sdk-1_3_0-linux.rpm in the current directory.
3. As a root user run the rpm command to install the packages:

   ```
   rpm -iv j2sdk-1_3_0-linux.rpm
   ```

   This will install the packages comprising the Java 2 SDK.

2.2.2 Java 2 Platform, Enterprise Edition 1.3 (J2EE)

Java 2 Platform, Enterprise Edition (J2EE) is a complete edition of the Java 2 platform that extends mission critical enterprise applications to any web browser[SUN2001a].
J2EE provides a component-based approach to the development and deployment of enterprise applications. The J2EE platform gives you a multi-tiered distributed application model, the ability to reuse components, a unified security model, and flexible transaction control. The developer can not only deliver customer solutions to market faster, but also make it platform-independent so that J2EE component-based solutions are not tied to the products and APIs of any one vendor.

The multi-tier J2EE applications are divided into the following three or four tiers as shown in Figure 2.2.

(1) Client tier: Application clients and HTML pages or applets are client components at the client tier.

(2) Web tier: JSP pages or servlets are web components at the web tier.

(3) Business tier: Enterprise beans are business tier at the business tier. They are used to solve or meet the needs of a particular business domain such as banking, retail, or finance.

(4) Enterprise information system tier: It includes enterprise infrastructure systems such as enterprise resource planning, database systems, and other legacy information systems.

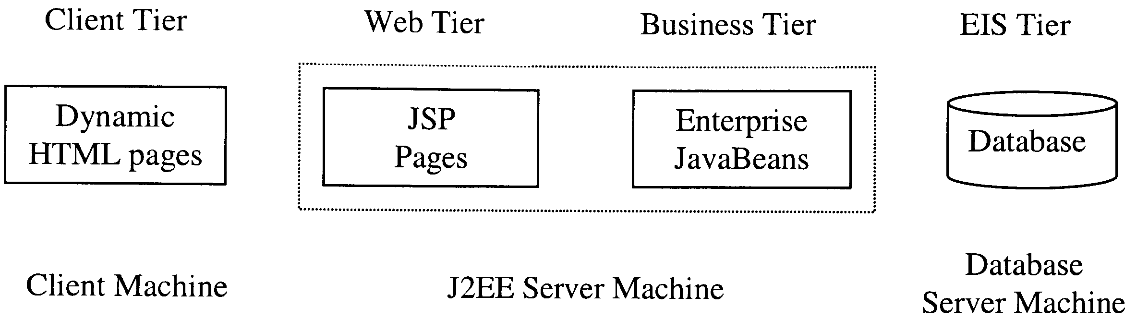


Figure 2-2 The Typical Multi Tiered J2EE Application

Installation procedure:

1. Before installation, you need to first install Java 2 SDK, Standard Edition (J2SE) on your machine.
2. Go to http://developer.java.sun.com/developer/earlyAccess/j2ee/ [SUN2001b]for downloading j2sdkee-1_3-beta-linux.tar.gz.
3. Change to the directory where you want to install the software. To uncompress and unpack the download bundle, run this command:

```
tar xvzf j2sdkee-1_3-beta-linux.tar.gz
```

The j2sdkee1.3 directory is created and the software is installed into it.

4. Edit the user configuration script that is in the userconfig.sh file of the bin directory of your installation. The userconfig.sh file sets these environment variables: J2EE_CLASSPATH - the classpath referenced by the J2EE server. It must include the location of JDBC driver classes. However, it does not need to include the Java 2 SDK, Enterprise Edition classes (j2ee-jar); the Java 2 SDK, Standard Edition software; or the classes contained in the enterprise application. JAVA_HOME - the absolute path of the directory in which the Java 2 SDK, Standard Edition is installed.
5. Update the PATH environment variable so it includes the bin directory of this release.

2.3 Tomcat Server 3.2.1

Tomcat is the Servlet+JSP Engine that is a subproject of the Jakarta Project, and it is the Reference Implementation for the Java Servlet 2.2 and JavaServer Pages 1.1[Jakarta1999]

Here are the major steps in the installation of Tomcat Server 3.2.1:

(1) Download jakarta-tomcat-3.2.1.zip from http://jakarta.apache.org/builds/jakarta-tomcat/release/v3.2.1/bin/

(2) Unzip the file into directory /usr/local. This should create a new subdirectory named "jakarta-tomcat-3.2.1".

(3) Change directory to "jakarta-tomcat-3.2.1" and set a new environment variable (TOMCAT_HOME) to point to the root directory of the Tomcat hierarchy as follows:
TOMCAT_HOME=/usr/local/jakarta-tomcat-3.2.1; export TOMCAT_HOME

(4) Set the environment variable JAVA_HOME to point to the root directory of your JDK hierarchy, then add the Java interpreter to your PATH environment variable. Usually this should have already been done when you install your Java platform.

(5) Start Tomcat using the following command:
/usr/local/jakarta-tomcat-3.2.1/bin/tomcat.sh start

2.4 MySQL 2.1.4

MySQL is one of the most popular open sources SQL database. It is provided by a company called MySQL AB. MySQL is a relational database management system. The tables are linked by defined relationships so it possible to combine data from several tables on request. MySQL is fast, reliable, and easy to use. The connectivity, speed, and security make MySQL highly suited for accessing databases on the Internet.

MySQL is a client/server system that consists of a multi-threaded SQL server that supports different back ends, several different client programs and libraries, administrative tools, and a programming interface.

If you install the Red Hat Linux 7 in a full set, MySQL should already have been installed, what you need to do is to find the location of the software and start the MySQL server (as root user) as follows:

```
/usr/bin/mysql_install_db
/usr/bin/safe_mysqld --user=mysql &
```

Then you can use /usr/bin/mysqladmin version to verify that the server is running.

Also, You can check to see the whole installation procedure by reading the MySQL manual [MySQL1995].

2.5 MySQL JDBC driver 2.0.4

The JDBC API defines Java classes to represent database connections, SQL statements, result sets, database metadata, etc. It allows a Java programmer to issue SQL statements and process the results. JDBC is the primary API for database access in Java. The JDBC API is implemented via a driver manager that can support multiple drivers connecting to different databases.

In a three tier model as shown in Figure 2.3, a user's commands are delivered to a "middle tier" of services, which then send SQL statements to the database. The database processes the SQL statements and sends the results back to the middle tier, which then sends them to the user.

You can download the MySQL JDBC driver 2.0.4 [Matthews2001] through http://mmmysql.sourceforge.net/.

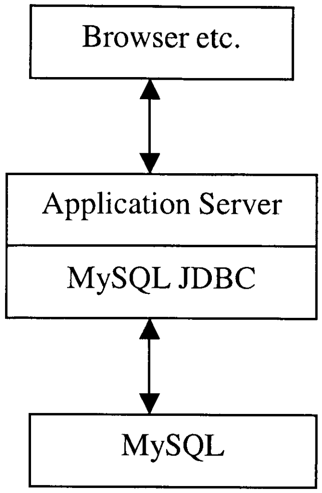


Figure 2-3 MySQL JDBC in a Three-tier Model

3 J2EE Platform Technologies

3.1 JDBC API

JDBC is a Java Application Programming Interface (API) that provides a standard framework for accessing data sources, usually relational databases that use the Structured Query Language (SQL). With the help of the JDBC API, it is possible to access wide ranges of SQL databases with the same Java syntax.

3.1.1 Goals for JDBC

Two goals are set for the design of JDBC[Asbury1999]:

(1) Support Common Database Standards

Because JDBC Supports common database standards, enterprise developers can take advantage of the latest Java technologies and at the same time can maintain a high degree of interoperability with other industry-standard products.

(2) Keep the API simple

Keep the common cases simple by providing standard activities with specific APIs. Application programmers can write and maintain less code with the methods for common cases being already provided. Actually one of the best features of JDBC is that Java programmers can develop database access strategies for their applications rapidly with the help of JDBC. A developer can create a connection to a database, query the database, and update values by writing very little code. JDBC also supports a transaction model so programmers can make updates in the database and undo all of them as a single transaction if it becomes necessary.

3.1.2 JDBC Architecture

The basic architecture of JDBC is not complicated. A class called DriverManager provides a service for managing a set of JDBC drivers. The DriverManager class attempts to load the driver classes referenced in the jdbc.drivers system property. Drivers can be loaded explicitly by using Class.forName. The driver should register itself with the DriverManager when it is loaded. The call DriverManager.getConnection() looks for a registered driver that can handle the data source described by the URL and returns an object from the driver. A connection represents a session with the data source and includes methods for executing database operations. Besides, a connection provides the so- called meta-data, which includes information about the data source structure. The protocol used to define the data source is in the format of jdbc:subprotocol:datasourcename. The getConnection() method takes username and password for registration with data sources, but the username and password may be omitted according to the design of the database.

In section 3.1.3 (Standard steps in querying databases), the architecture described above will be explained in more detail.

According to the location of the database, the driver, the application, and the communication protocols used, there are several typical scenarios for using JDBC, which can be grouped into two- and three-tier architectures.

In a two-tier architecture, the application and the database driver are on the same machine. The JDBC driver can access the database, which is running on a database server through network. As such, the database driver is responsible for handling the networked communication. A simple two-tier JDBC architecture is shown in Figure 3.1. As we can see from the figure, the Java application running on the client machine uses a JDBC driver that resides on the same machine. The local driver uses a vendor-specific client library for accessing the database remotely through the network. The Java application accesses this resource transparently with no need to deal with network communication issues by itself.

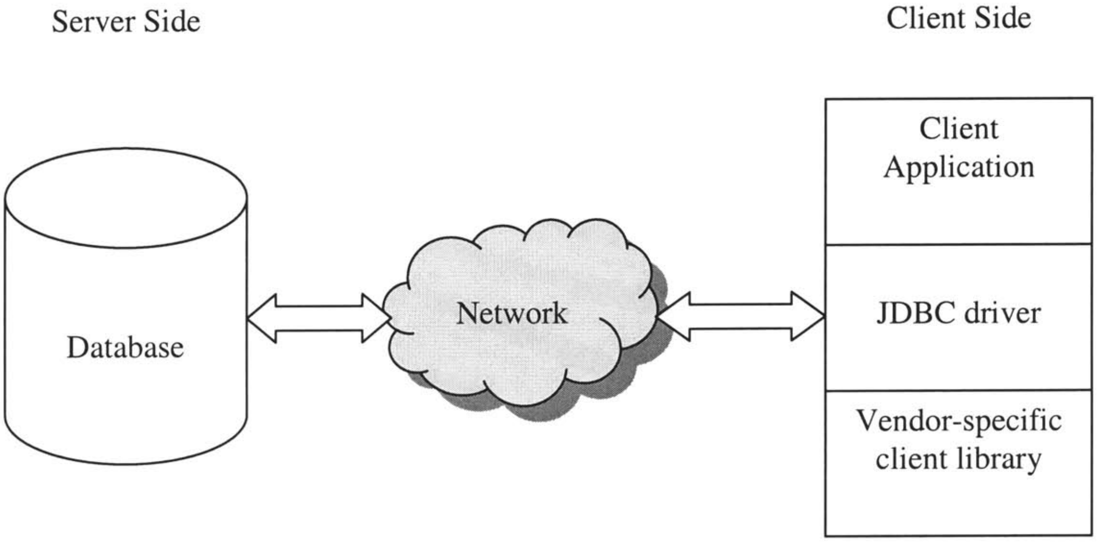


Figure 3-1 A Two-tier JDBC Architecture.

In a three-tier architecture, the application or applet runs on one platform and the database driver is located on another. The database driver can be accessed through the following mechanisms:

- An applet may access the driver through a Web server
- An application may access a remote server program that communicates locally with a database driver
- An application may communicate with an application server that accesses the database on behalf of an application

A three-tier architecture is shown in Figure 3.2 in which an applet on the client Web browser communicates with a server application and the server application accesses a database behind the firewall.

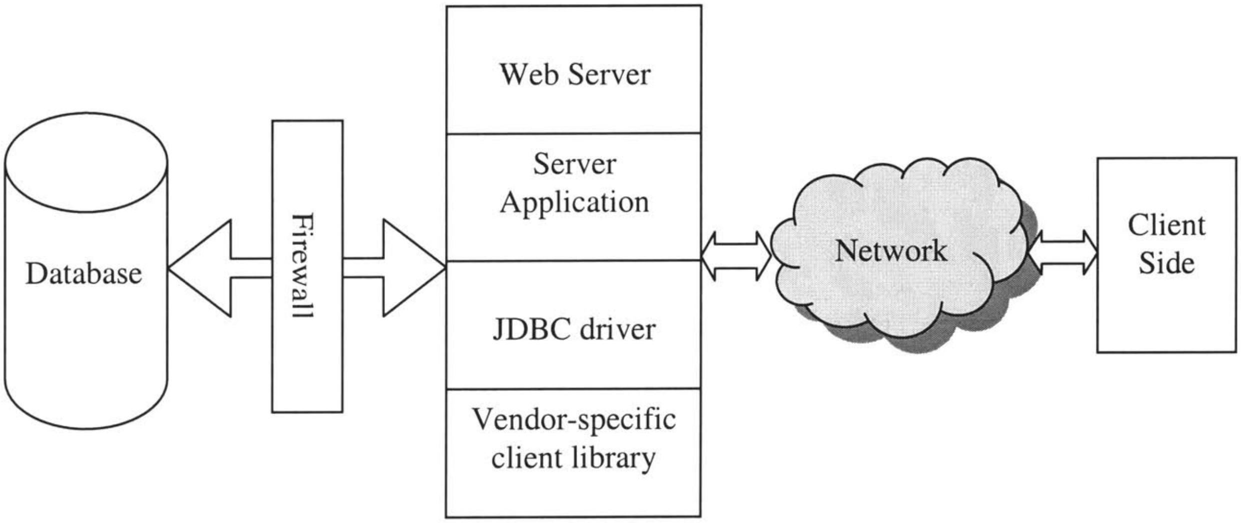


Figure 3-2 A Three-tier JDBC Architecture.

By encapsulating all the database access logic in the JDBC driver, the driver vendor deals with the issues of communicating with the database and database vendor's client library. Therefore we can write applications that function in a two-tier or three-tier environment with few or no changes to the code. The JDBC design allows any Java programmer to access a relational database with little additional effort.

3.1.3 Standard steps in querying databases

The following standard steps are needed in order to query databases[Hall2000]:

(1) Load the JDBC driver

The JDBC driver is a program that knows how to communicate with the actual database server; it translates calls written in the Java programming language into the specific format required by the server. In order to load the driver, we only need to use Class.forName as follows:

```
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
```

The above segment is used to load MySQL driver. For other drivers, the Class.forName statement should be changed accordingly. For example, the statement corresponding to the Oracle and Sybase driver are as follows:

```
Class.forName("oracle.jdbc.driver.OracleDriver”);
Class.forName(“com.sybase.jdbc.SybDriver”);
```

(2) Establish the connection

Once the JDBC driver is loaded, we need to pass the URL, the database username and password to the getConnection method of the DriverManager class in order to make the actual network connection. The URL uses the jdb :protocol and the server host and database name are embedded with in the URL. For MySQL database, the statements to establish the connection are shown as follows:

```
dbCon = DriverManager.getConnection(
            "jdbc:mysql://localhost/test_pavement?user=&password=");
```

(3) Create a statement object

After establishing the connection, now we can create a statement object from the connection as follows:

```
Statement statement =  dbCon.createStatement ();
```

The statement object is used to send queries and commands to the database.

(4) Execute a query

After we get a statement object, we can use the object to send SQL queries by using the executeQuery method. This method returns an object of type ResultSet as follows:

```
String query = "SELECT * FROM Holiday";
ResultSet rs = statement.executeQuery(query);
```

(5) Process the results

After we get the result set, we can use several methods such as next (move through the table a row at a time), get Xxx (take a column index or column name as an argument and return the result as a variety of different Java types), findColumn (get the index of the named column), wasNull (was the last getXxx result SQL NULL?) and getMetaData (retrieve information about the ResultSet in a ResultSetMetaData object) to process the results.

For example, the following statements print out the first column in all rows of a ResultSet.

```
While (rs.next()) {
   System.out.println(rs.getString(1));
}
```

(6) Close the connection

We simply use the following statement to close the connection:

```
dbCon.close();
```

3.2 Web Components

A Web component is a software application that provides a response to a request[SUN2001a]. A Web component usually generates the user interface for a Web-based application. There are two kinds of Web components: servlets and JavaServer Pages (JSP).

3.2.1 Web Component Roles

In the J2EE application programming model Web components can serve two roles: as front components and as presentation components.

(1) Front Components

Instead of doing presentation, front components manage other components and handle HTTP requests or convert the requests into a form that an application can understand. Front components provide a single entry point to an application, thus making security, application state, and presentation uniform and easier to maintain.

(2) Presentation Components

Presentation components generate the HTML/XML response that can determine the user interface. A JSP page acting as a presentation component may contain presentation logic or reusable custom tags. A custom tag is a user-defined JSP language element that increases productivity by encapsulating recurring tasks so that they can be reused across more than one application. Besides, a servlet can also be a presentation component.

Modular design facilitates separation of roles. Content providers can concentrate on how content is displayed, and component developers can focus on the logic that is used in the JavaBeans component to manipulate the data, and on the JSP page that generates the HTML representation of the data.

Other requirements that presentation components must address are creating a consistent look and feel for an application and at the same time providing mechanisms for personalizing the user interface. For example, consider an application in which all pages share a common banner, navigation menu, body, and footer as shown in Figure 3.3. Each item in the example can be seen as a component that is used to generate the final look and feel, can contain dynamic information, and should be customizable.

There are two ways of constructing the page shown in Figure 3.3. We could build the page using custom tags and JavaBeans components or break up each portion into separate JSP pages each containing their own necessary custom tags and JavaBeans components, then build the whole page from a JSP page that integrates the others using runtime includes.

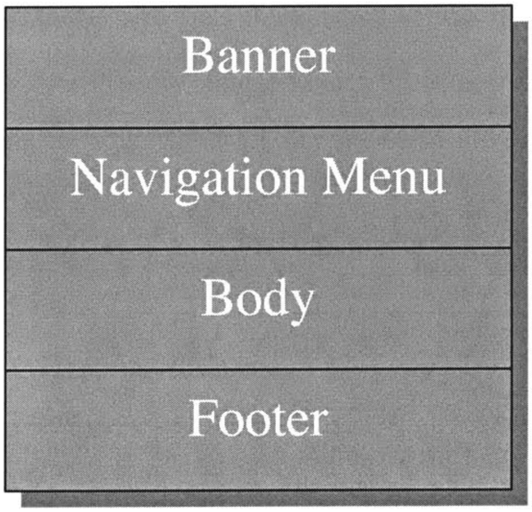


Figure 3-3 Presentation Components

When appropriately arranged, a clean separation between presentation logic, data, and content can be achieved.

It is a good practice to use JavaBeans components or custom tags to do data rendering. These are created by a developer who is familiar with the Java programming language. If JavaBeans components and custom tags are designed in a general manner, they should be reusable in other parts of the application or in other applications.

3.2.2 Servlets

A servlet is a program that extends the functionality of a Web server. Servlets receive a request from a client, generate the response on the fly and then send the response containing an HTML or XML document to the client.

A servlet developer uses the servlet API to:

- Initialize and finalize a servlet
- Access a servlet's environment
- Receive requests and send responses
- Maintain session information on behalf of a client
- Interact with other servlets and other components

Usually JSP pages can be used for most purposes. However, under some special circumstances, servlets are more appropriate. The following are these two circumstances.

Generating Binary Data

Servlets are well suited for dynamically generating binary data such as images or a new content type. Requests for content of that type would be mapped to servlets that know how to generate the content, but from the Web client's point of view, it is merely requesting delivery of an ordinary image.

Extending a Web Server's Functionality

Servlets are a portable mechanism for extending the functionality of a Web server. For example, if a new data format must be supported, a servlet can be mapped to the file type for the format.

A good example of a servlet that extends a Web server is the servlet that is mapped to JSP files. This servlet parses all files that end with a jsp file extension and compiles the JSP pages into servlets. The Web container then executes the resulting servlets and the resulting response is sent back to the client.

3.2.3 JavaServer Pages Technology

The JavaServer Pages (JSP) technology provides a new way to generate dynamic content for a Web client. A JSP page is a text-based document that describes how to process a request to create a response. A JSP page usually contains:

- Template data to format the Web document. Typically the template data uses HTML or XML elements. Document designers can edit and work with these elements on the JSP page without affecting the dynamic content. This makes development simpler because it separates presentation from dynamic content generation.

- JSP elements and scriptlets to generate the dynamic content in the Web document. Most JSP pages use JavaBeans and/or Enterprise JavaBeans components to perform the more complex processing required of the application.

- Standard JSP actions can access and instantiate beans, set or retrieve bean attributes, and download applets. JSP is extensible through the development of custom actions. The custom actions are encapsulated in tag libraries.

3.2.4 JSP Pages Versus Servlets

Both JSP pages and servlets have their own advantages thus should be used according to the actual situation. In some cases it is hard to say whether a servlet or JSP page is better. When both servlet and JSP technology is available in an environment, JSP pages should usually be used. However, servlets are best suited for low-level application functions that don't require frequent modification.

The Java code used within JSP pages should remain relatively simple. Therefore, we should encapsulate complex tasks within custom tags and JavaBeans components. Even a sophisticated Web application can consist solely of JSP pages, custom tags, and JavaBeans components; servlets are generally not necessary.

JSP pages combine dynamic content with logic in a presentation-centric, declarative way. JSP pages should be used to handle the HTML representation that is generated by a page. They are coded in HTML-like pages with structure and content familiar to Web content providers. However, JSP pages are more powerful than ordinary HTML pages. JSP pages can handle application logic through the use of JavaBeans components and custom tags. JSP pages themselves can also be used as modular, reusable presentation components that can be bound together using a templating mechanism.

3.2.5 Web Component Containers

Servlet containers, JSP containers, and Web containers host web components.

In addition to standard container services, a servlet container provides network services, decodes requests, and formats responses. All servlet containers must support HTTP as a protocol for requests and responses, but may also support additional request-response protocols such as HTTPS.

A JSP container provides the same services as a servlet container and an engine that interprets and processes a JSP page into a servlet.

A Web container provides the same services as a JSP container and access to the J2EE service and communication APIs.

3.3 EJB Components

The Enterprise JavaBeans (EJB) component architecture is a reusable, server-side component model that enables enterprises to build scalable, secure, cross-platform, mission-critical applications[Roth1998]. Because the EJB developers no longer need to write code that deals with transaction management, security, connection pooling etc, they can focus their attention on writing business logic.

3.3.1 The Advantages of EJB Technology

The Enterprise JavaBeans Technology has the following advantages:

- High productivity: The developers gain productivity by developing on the Java platform. More importantly, as mentioned above, they gain productivity by focusing only on writing their own business logic.

- Industry Support: Customers who are going to build EJB systems have a lot of solutions to choose from. Dozens of companies are supporting or are going to support Enterprise JavaBeans technology.

- Protection of Investments: Enterprise JavaBeans technology builds on top of the existing systems in the enterprise today. So the investment made by an enterprise will be protected.

- Architectural Independence: Because Enterprise JavaBeans technology insulates the developer from the underlying middleware; any improvement and change in the middleware layer made by EJB server vendor will not affect a user's EJB application.

- Server-Side Write Once, Run Anywhere: EJB technology takes the notion of Java Platform's Write Once, Run Anywhere one step further to run on any server that strictly implements the Enterprise JavaBeans APIs.

3.3.2 Enterprise Application Models

EJB technology provides an infrastructure that take care of system level programming such as transactions, security, threading, naming, object-life cycle, resource pooling, remote access, and persistence etc so those originally manually coded features become simple declarative properties of the enterprise beans. Therefore the effort to build distributed applications is greatly reduced and high development efficiency is achieved.

There are two enterprise application models and EJB covers both models because it is designed to be broadly applicable.

For one model, the client begins a session with an object that performs a task on behalf of the client and session beans cover this model. A session bean is an object that represents a transient conversation with a client, and its fields contain the state of the conversation and are transient. For this reason, a session bean is gone if either the server or the client crashes.

For the other model, the client accesses an object that represents an entity in a database and entity beans cover this model. An entity bean represents data in a database, as well as the methods to act on that data. Entity beans are transactional, and are persistent. As long as the data remains in the database, the entity Bean exists.

3.3.3 EJB Architecture

Figure 3.4 shows the architecture of EJB technology.

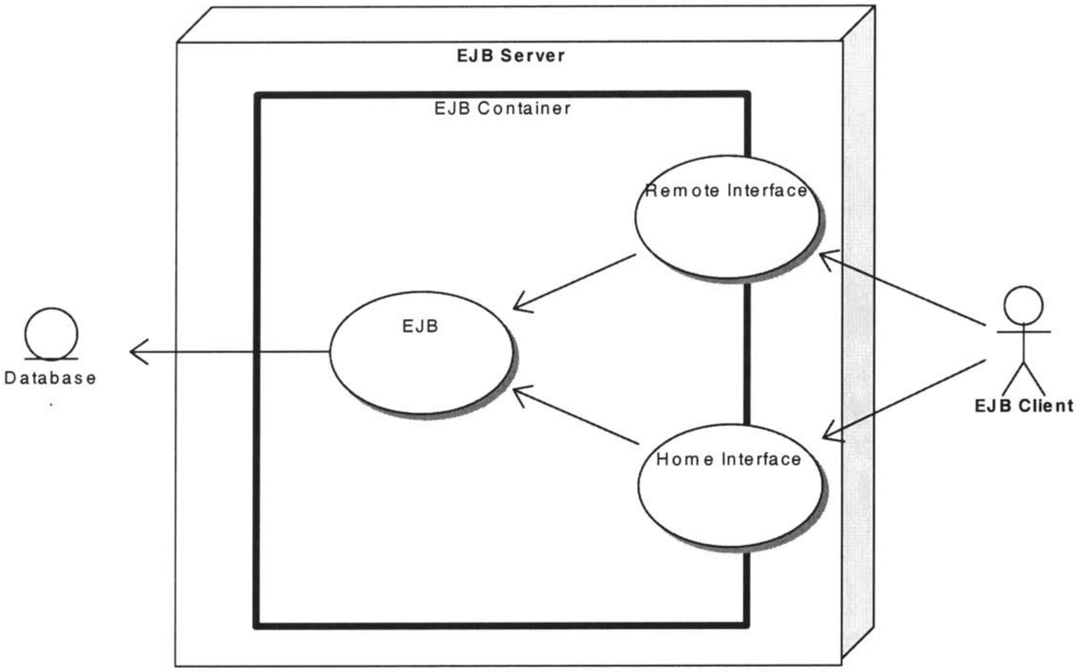


Figure 3-4 EJB Architecture

The EJB server is a collection of services for supporting an EJB installation and it manages the resources needed to support EJB components.

The EJB container is where an Enterprise Bean lives, just as a database is where a record lives. It provides a scalable, secure, transactional environment in which Enterprise Beans can operate. It handles the creation and destruction of an object.

When an Enterprise Bean is installed in a container, the container provides an implementation of the bean's EJBHome interface, and remote interface. It also makes the bean's EJBHome interface available in the Java Naming and Directory Interface (JNDI).

The remote interface is automatically generated when an Enterprise Bean is installed on a server. The EJBObject, which is an implementation of the remote interface, acts like a proxy, taking the remote object invocations and calling the appropriate methods on the Enterprise Bean instance.

The EJBHome interface is implemented when an Enterprise Bean is installed in the container. The EJBHome interface allows for the creation and deletion of a bean as well as querying information about a bean.

3.3.4 EJB Developer Roles

Five developer roles are existed in EJB technology. They are server provider, container provider, Enterprise Beans provider, application assemblers, and deployers.

Server providers specialize in distributed transaction management. They deal with distributed objects and low-level systems services.

Container providers specialize in systems programming. The container provides the secure, scalable, transactional environment for a bean, so the provider needs experience in these areas.

Enterprise Bean providers provide the components for EJB applications. They are typically domain experts developing business logic in the form of beans. They include their components into an EJB JAR file.

Application assemblers build applications from third party Beans. They are domain experts and may also build a GUI on the client side.

Deployers are usually familiar with an enterprise's operational environment. They take packaged applications and set the applications' security and transaction descriptors.

4 Database Design for the Arlington Permit System

In this chapter we will describe the database design for the Arlington permit system.

‘

4.1 Entity-Relationship Model

An Entity-Relationship (E-R) diagram depicts entities that represent things or concepts and the relationships that represent associations between each pair of entities. In this section, as our first step in database design, an Entity-Relationship model is employed to construct the concept design of the database. Entities are identified and relations are established.

Although some experts suggest that attributes should be included in the E-R model by putting them in ovals and connecting them by lines with the entity, some others recommend keeping attributes out of the E-R diagram[Gilbert1998].

Because some entities in our database have many attributes, including them in the E-R diagram may make the E-R diagram difficult to read, we decided not to include the attributes in our E-R diagram. Instead, in order to keep track of all these attributes, we will list them in next section when we begin to analyze normalization and denormalization.

The E-R diagram is shown in Figure 4-1.

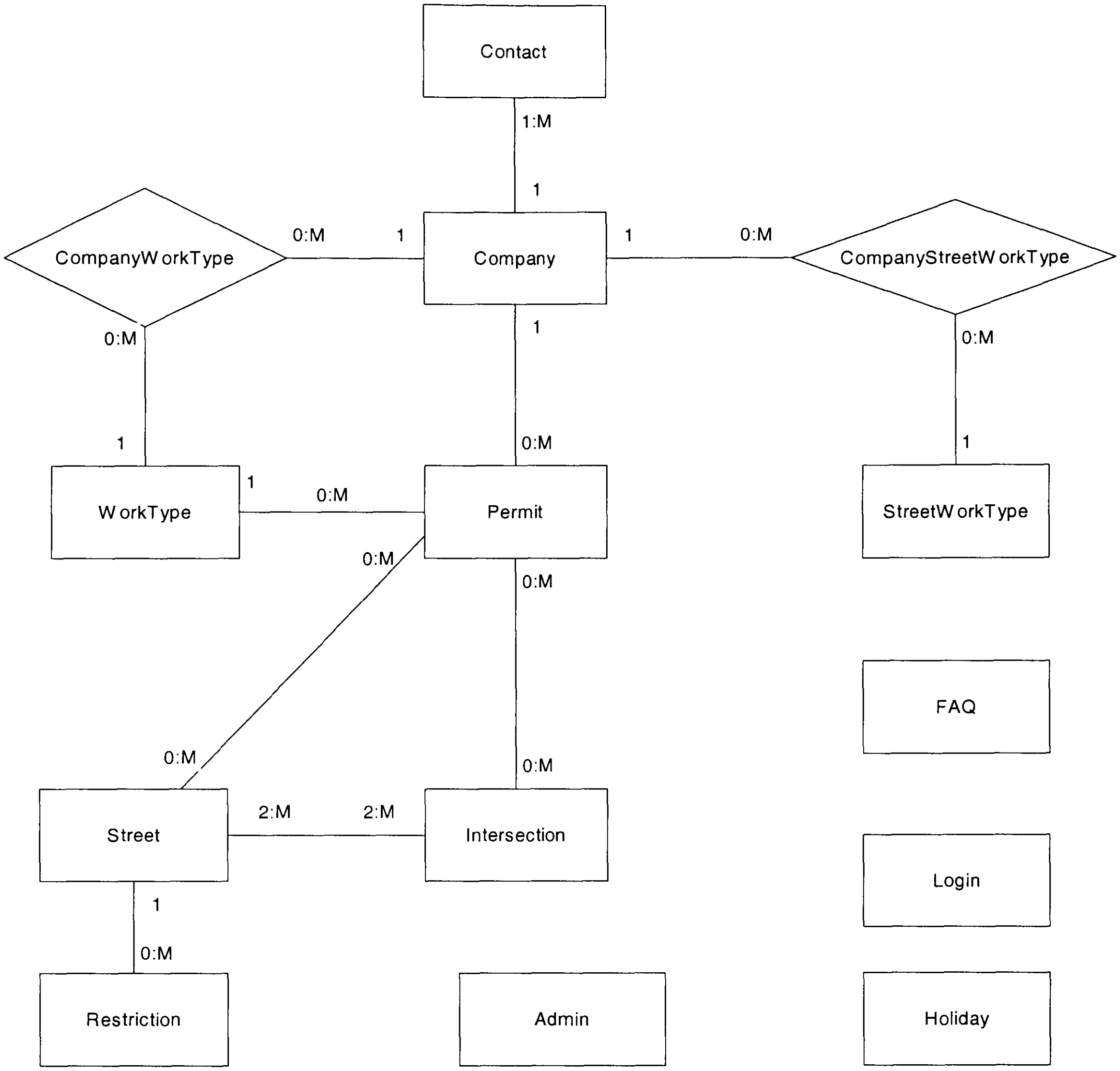


Figure 4-1 E-R Diagrams for Arlington Permit System

4.2 Normalization and Denormalization

In this section we will list the attributes for all tables that we used in our system and evaluate the tables according to normalization and denormalization rules.

Before we actually evaluate each table, let us take a look at normalization and denormalization.

The objective of normalization is mainly to prevent the occurrence of update anomalies through proper decomposition of relations.

There are two grouping of normal forms[Elmasri1999]: The first group includes First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and Boyce-Codd Normal Form (BCNF). The second group includes Fourth Normal Form (4NF) and Fifth Normal Form (5NF).

Because normalization to 5NF is not usually necessary from a practical point of view, normalization to BCNF should be enough. In this section, we only restrict our discussion of normalization to BCNF.

1NF: A relation is said to be in 1NF if and only if all underlying domains contain atomic values only. This requires a relation can neither have a multi-valued attribute nor a composite attribute.

2NF: A relation is said to be in 2NF if and only if it is in 1NF and every non-key attribute is not partially dependent on any key of the relation. This requires us to take out attributes in the relations that are dependent only on a part of the key.

3NF: A relation is said to be in 3NF if and only if it is in 2NF and every non-key attribute is non-transitively dependent on every key of the relation.

BCNF: A relation is said to be in BCNF if and only if every determinant that fully determines some other attributes in the relation is a key. This requires us to take out all attributes in the relations that do not solely dependent on the key.

Denormalization is an approach to help balance the goals (ease of use, performance, maintainability) and limitations (development time, limited machine power, program complexity). In the past it was usually important to denormalize after complete normalization in order to achieve a better performance and to make the application easier to use. But now the situation is changing greatly. Especially in our model, denormalization is never really necessary since the database is small and performance would be good with a fully normalized model. So we will not discuss denormalization further more and only limit the discussion to normalization.

Now let us evaluate each table in our system one by one.

Table 4-1 Admin Table

Field Name	Definition
AdmBasicFee	The basic application fee for a street opening in US dollar (e.g. $25).
AdmFrom1	These fields provide information about additional fees for utilities (Water) according to the length of street work area. The format is as follows: From work area length of *AdmFrom1* feet to *AdmTo1* feet, the additional fee is *$ AdmAddFee1*; From work area length of *AdmFrom2* feet to *AdmTo2* feet, the additional fee is *$ AdmAddFee2*; From work area length of *AdmFrom3* feet to *AdmTo3* feet, the additional fee is *$ AdmAddFee3.*
AdmTo1	
AdmAddFee1	
AdmFrom2	
AdmTo2	
AdmAddFee2	
AdmFrom3	
AdmTo3	
AdmAddFee3	
AdmIssueWeekend	This flag determines whether permits are issued on weekends or not.
AdmIssueHolidays	This flag determines whether permits are issued on holidays or not.
AdmWeekendFrom1	This field determines what day of the week the weekend starts.

AdmWeekendFrom2	This field determines what time of the day the weekend starts.
AdmWeekendFrom3	This field determines whether the weekend starts from the morning (am) or the afternoon (pm).
AdmWeekendTo1	This field determines what day of the week the weekend ends.
AdmWeekendTo2	This field determines what time of the day the weekend ends.
AdmWeekendTo3	This field determines whether the weekend ends from the morning (am) or the afternoon (pm).
AdmMinNotification	The minimum notification time before work may commence in hours (e.g. 48 hours)
AdmMaximumValidDays	Maximum valid days for a permit
AdmWorkTypeChecking	The flag to determine whether to perform company work type checking or not.
AdmStreetWorkTypeChecking	The flag to determine whether to perform street work type checking or not
AdmPremisesChecking	The flag to determine whether to perform premises range checking or not
AdmDrawingFileRequired	The flag to determine whether to require drawing file from utilities or not
AdmEmailNotification	The flag to determine whether to send email notification for each issued permit or not
AdmMailServer	SMTP Server for email notification
AdmEmailAddress	Email Address for email notification

This table has many attributes. These attributes are all single atomic values, so it is in 1NF. But the following functional dependencies are identified:

{ AdmFrom1, AdmTo1} → AdmAddFee1;

{ AdmFrom2, AdmTo2} → AdmAddFee2;

{ AdmFrom3, AdmTo3} → AdmAddFee3;

In other words, AdmAddFee1, AdmAddFee2, AdmAddFee3 are not fully dependent on the all the keys in the relation, so it is not in 2NF. Besides, they are also repeating groups and should be put in a separate table. But as we know, this table stores all miscellaneous options and has only one record. The attributes directly correspond to one web page, and our permit system is simple enough, so from a practical point of view, it is fine to put all these attributes in one table.

Table 4-2 Company Table

Field Name	Definition
ComID	The company ID uniquely identifies a contractor. It is an internal parameter used to reference the full contractor name. [Primary key]
ComName	The company name. Can be either a contractor name or a utility company name.
ComType	The type of the company. Can be either ’utility’ or ’contractor’.
ComContact	A contact name. It is optional and may be useful for large companies.
ComLicense	Contractors have a license number, which is also used as the password for the contractor login.
ComStreet	Company’s street name.
ComCity	Company’s city.
ComState	Company’s state.
ComZip	Company’s zip code.
ComPhone	Company’s phone number.
ComFax	Company’s fax number.
ComPager	Company’s pager number.
ComEmail	For utility companies, the email is also used as the password.
ComActive	The state of a company. Inactive Companies will not be listed in outputs unless specifically requested, and will not be allowed to apply for permits.

In the Arlington Permit System, a company often has several contact persons for permit application. Besides, there are several attributes such as contact name, contact phone and contact email etc. So ComContact is a multi-valued attribute as well as a composite attribute. We have to create a separate relation that holds the primary key (ComID) and the attributes of that composite attribute. This table is already been created by the name of Contact. So in order to make the Company table in 1NF, we should delete the ComContact attribute.

ComStreet, ComCity and ComState together can determin ComZip, so from this point of view, it does not satisfy the 3NF rule. We could delete ComZip from the table and add another table that list Street, City, State and its corresponding zip code, but in our system it may not necessary. Actually zip codes are the most common example of violating 3NF, yet people usually allow this special case of violating 3NF.

Table 4-3 CompanyStreetWorkType Table

Field Name	Definition
CSTCompanyID	The company ID from the company table. [Primary key, foreign key]
CSTType	The type of street work the company performs, according to the StreetWorkType table. [Primary key, foreign key]

This table is an intermediate table and it is very simple. It is in BCNF.

Table 4-4 CompanyWorkType Table

Field Name	Definition
CWTCompanyID	The company ID from the company table. [Primary key, foreign key]
CWTType	The type of work the company performs, according to the WorkType table. [Primary key, foreign key]

This table is also an intermediate table and very simple. It is in BCNF.

Table 4-5 Contact Table

Field Name	Definition
ConCompanyID	The company ID. [Primary key, foreign key]
ConName	Contact name. [Primary key]
ConPhone	Phone number of the contact.
ConPager	Pager number of the contact.
ConEmail	Email address of the contact.

It is in BCNF.

Table 4-6 FAQ Table

Field Name	Definition
FAQID	Sequentially assigned number of the frequently asked question. [Primary key]
FAQQuestion	The question text.
FAQAnswer	The answer text.
FAQPosition	The position of that question and answer set located.

It is in BCNF.

Table 4-7 Holiday Table

Field Name	Definition
HolName	The name of the holiday. [Primary key]
HolDate	The date of the holiday.

It is in BCNF.

Table 4-8 Intersection Table

Field Name	Definition
IntID	The intersection ID number will uniquely identify an intersection. [Primary key]
IntGoverningStreetID	The ID number of the governing street of the intersection, usually the major one of the intersecting streets.
IntSecondStreetID	The ID number of the second street of the intersection.
IntClosestPremises	The closest premises to the intersection on the governing street. This field is required to check against restrictions and to map the openings added to the street.

It is in BCNF.

Table 4-9 Login Table

Field Name	Definition
LogName	The user names (internal users only). [Primary key]
LogPassword	The password of that user.
First_Name	The first name of that user.
Last_Name	The last name of that user.
Department	The department of that user.
Position	The position of that user.
Email	The email address of that user.

The attributes of this table are all single atomic values, so it is in 1NF. But the following functional dependencies are identified:

{ First Name, Last Name} → Department;

{ First Name, Last Name} → Position;

In other words, Department and Position are not fully dependent on the all the keys in the relation, so it is not in 2NF. So it is suggested to create a separate table with LogName, FirstName, LastName, Department, and Position as its attributes. But just to keep the system simple, we still put them in the same table.

Table 4-10 Permit Table

Field Name	Definition
PerID	Permit ID number. Each permit has a unique ID. [Primary key]
PerIssue	The issue date of the permit.
PerComID	The company ID number. This number identifies the company in the company table. [Foreign key]
PerWorkType	The type of work, according to the types defined in the WorkType table. [Foreign key]
PerDigSafe	The digSafe-number as entered by the applicant.
PerContact	The contact name. It is the name entered by the applicant. This will ensure that the person responsible for the application can be identified.
PerApproved	The application may be approved or not approved. This is determined with this flag.
PerValidFrom	The day when the street work may commence.
PerValidUntil	The last day when work may be conducted.
PerStreetWorkType	The type of street work, according to the types defined in the StreetWorkType table. [Foreign key]

PerStreetID	The street ID. This number identifies the street in the street table. [Foreign key]
PerIntersection	This flag determines whether the application was made for an intersection.
PerStartPremises	The premises number in front of which the work area starts.
PerEndPremises	The premises number in front of which the work area ends.
PerIntersectingStreetID	The street ID that intersecting with the other street to form an intersection. This street number and the other street number together identify the intersection where the work is conducted. [Foreign key]
PerWorkAreaLength	The length of the work area.
PerWorkAreaWidth	The width of the work area.
PerPurpose	Description of the purpose of work. Right now, this is a text field that has a limit of 255 chars.
PerDrawingFile	Name of the drawing file that is required for utility companies
PerFileContentType	File content type for the drawing file that is required for utility companies
PerFileLength	File length of the drawing file that is required for utility companies
PerUtilityEmail	Email address for the utility company
PerApplicationFee	The total fee assessed to this application in US-$.
PerTelephoneManholes	Number of telephone manholes added to the street section(s) within the work area of this permit.
PerWaterManholes	Number of water manholes added to the street section(s) within the work area of this permit.
PerSewerManholes	Number of sewer manholes added to the street section(s) within the work area of this permit.
PerElectricManholes	Number of electric manholes added to the street section(s) within the work area of this permit.

PerCatchBasins	Number of catch basins added to the street section(s) within the work area of this permit.
PerWaterGates	Number of water gates added to the street section(s) within the work area of this permit.
PerGasGates	Number of gas gates added to the street section(s) within the work area of this permit.
PerOtherOpenings	Number of other openings added to the street section(s) within the work area of this permit.
PerTimeStamp	This additional field is used to track the submission of new manhole data to the pavement management system.
PerActive	To remove an application, its active parameter is set to No, which will remove this permit from all reports and views.
PerLastEditDate	This field indicates the date when the permit was last modified.
PerLastEditUser	The name of the user who was the last modifying the permit.

This table has many attributes, but its attributes are all single atomic values, so it is in 1NF. But the following functional dependencies are identified:

PerWorkAreaLength→PerApplicationFee

In other words, PerApplicationFee is transitively dependent on the all the keys in the relation, so it is not in 3NF. Actually PerApplicationFee is a derived attribute and might be deleted from the table. But for the sake of simplicity, it is fine to have the derived attribute in the table.

Table 4-11 Restriction Table

Field Name	Definition
ResID	The ID number of the restriction. This is required only internally to uniquely identify each restriction. [Primary key]

ResStreetID	Identifies the street that is restricted in the Street table. [Foreign key]
ResWholeStreet	This flag determines whether the whole street or just a section of the street is restricted.
ResStartPremises	The start premises from where the street is restricted.
ResEndPremises	The end premises to where the street is restricted.
ResStartDate	The first date (included), from when the street is restricted.
ResEndDate	The last date (included), to when the street is restricted.
ResComment	Comment on the reason why the restriction was applied.
ResActive	Determines whether the restriction is currently active or not. Restrictions are "deactivated", but not deleted to maintain a history of restrictions.

It is in BCNF.

Table 4-12 Street Table

Field Name	Definition
StrID	The Street ID uniquely identifies each street. [Primary key]
StrName	The name of the street.
StrRemarks	Any remarks regarding the street

It is in BCNF.

Table 4-13 StreetWorkType Table

Field Name	Definition
SWTType	Types of street work. Values are ’Street opening only’, ’Sewer only’, ’Water only’, ’Sewer and water’, ’Other’

This is a simple validation table. It is in BCNF.

Table 4-14 WorkType Table

Field Name	Definition
WoTType	Pre-defined types of work. Values are 'Gas’, 'Electric’, 'Telco’, 'CableTV’, 'Drain/Water/Sewer’, 'Driveway’, 'Other’

This is a simple validation table. It is in BCNF.

From the discussion above, we can see that most tables are in BCNF. Some tables could be normalized, but because our system is simple and time was limited, the simplifications noted above were made.

5 Case Study of the Online Street Construction Permit System-An Enterprise Java Beans Approach

In this chapter, we will use the ProcessPermit session bean and the Permit entity bean as examples to demonstrate how the Enterprise JavaBeans Technology can be used in the online street construction permit system. First we will discuss Enterprise JavaBeans Technology in the context of the online street construction permit system, and then a step-by-step procedure of deploying EJB will be illustrated [Bodoff2001].

5.1 Enterprise JavaBeans Technology

The Enterprise Java Beans specification specifies the architecture for the development and deployment of transactional server-side software components that are called enterprise beans. These enterprise beans are distributed objects that reside in Enterprise JavaBeans containers. The distributed clients throughout the network receive the remote services provided by the enterprise beans.

The EJB specification separates the distributed computing infrastructure from the business logic by the EJB API, which defines protocols, classes and interfaces. These specifications ensure portability across vendors.

5.1.1 The EJB Container

As mentioned earlier, the EJB container is an environment that hosts and manages Enterprise beans. An enterprise bean must work in this environment; otherwise it cannot function. The EJB container manages all the major aspects of an enterprise bean such as remote access to the bean, security, persistence, transactions, concurrency, and access to and pooling of resources.

The container shields the client from accessing the enterprise bean directly, so the client has to communicate with the enterprise beans via the container. This arrangement ensures that the container performs all the management of persistence, transactions, and security, so the bean developer can concentrate on the implementation of business logic without any need to take care of anything else.

Containers manage many beans at the same time. In order to reduce memory consumption and processing, they manage the beans and pool resources carefully.

An enterprise bean relies on the container to do a lot of things, from accessing a JDBC connection or another enterprise bean, to accessing the identity of its caller or obtaining a reference to itself. The enterprise bean uses one of the three mechanisms to interact with its container.

- Callback methods
- EJBContext interface
- Java Naming and Directory Interface (JNDI)

The callback methods are defined by the Enterprise Bean interface. Every enterprise bean implements a subtype of that interface. The container invokes these methods to notify the bean when it's going to activate the bean, end a transaction, remove the bean from memory, etc. In this way, it gives the bean a chance to perform some action before or after some event.

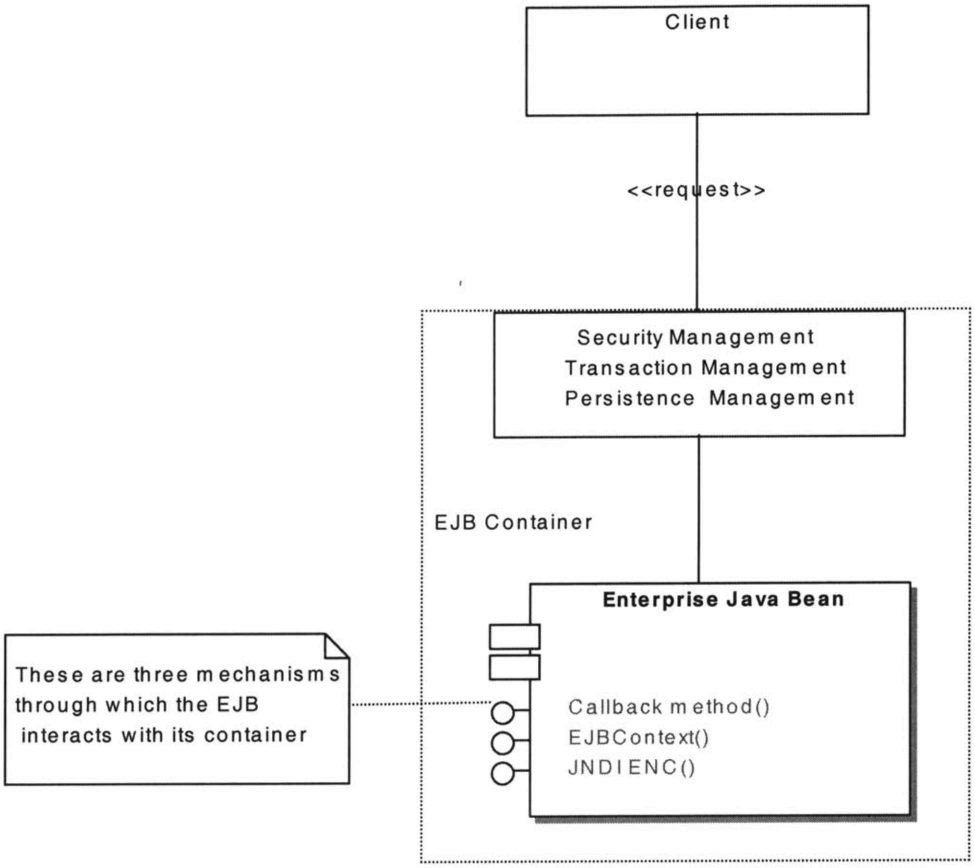


Figure 5-1 EJB Container

The EJBContext interface provides methods to communicate with the container so that the enterprise bean can obtain information about its environment like the identity of its client and the status of a transaction etc.

The Java Naming and Directory Interface (JNDI) is an extension to the Java platform for accessing naming systems like LDAP and file systems, etc. Every enterprise bean has access to a special naming system called the Environment Naming Context (ENC), which is managed by the container and accessed by enterprise beans using JNDI. The JNDI ENC allows a bean to access resources like JDBC connections and other enterprise beans etc.

The EJB specification defines a bean-container protocol that is designed to make enterprise beans portable between EJB containers so that enterprise beans can be developed once and run in any EJB container.

Besides the advantage of portability, the Enterprise Java Beans makes the programming model simpler. The developers are relieved from the tedious task of managing security, transactions, concurrency and resources etc. Instead, the developer only needs to concentrate his or her attention on the business logic.

5.1.2 Enterprise Beans

EJB server-side components include remote and home interfaces that define a bean's business methods, as well as the bean class that implements the interfaces. The client accesses the enterprise beans over the network through their home and remote interfaces. These interfaces define the methods that are used to create, update and delete the bean from EJB server. A bean is a server-side component that represents a business concept like a permit or a permit processor. This concept is shown in Figure 5-2.

5.1.2.1 Remote and Home Interfaces

Although both the remote and home interfaces represent the bean, the home interface defines the life-cycle methods of the component such as create, destroy, find etc. while the remote interface defines the business methods of the bean such as to use theProcessPermit bean to calculate the permit fee. The remote interface extends the javax.ejb.EJBObject interface while the home interface extends the javax.ejb.EJBHome interface.

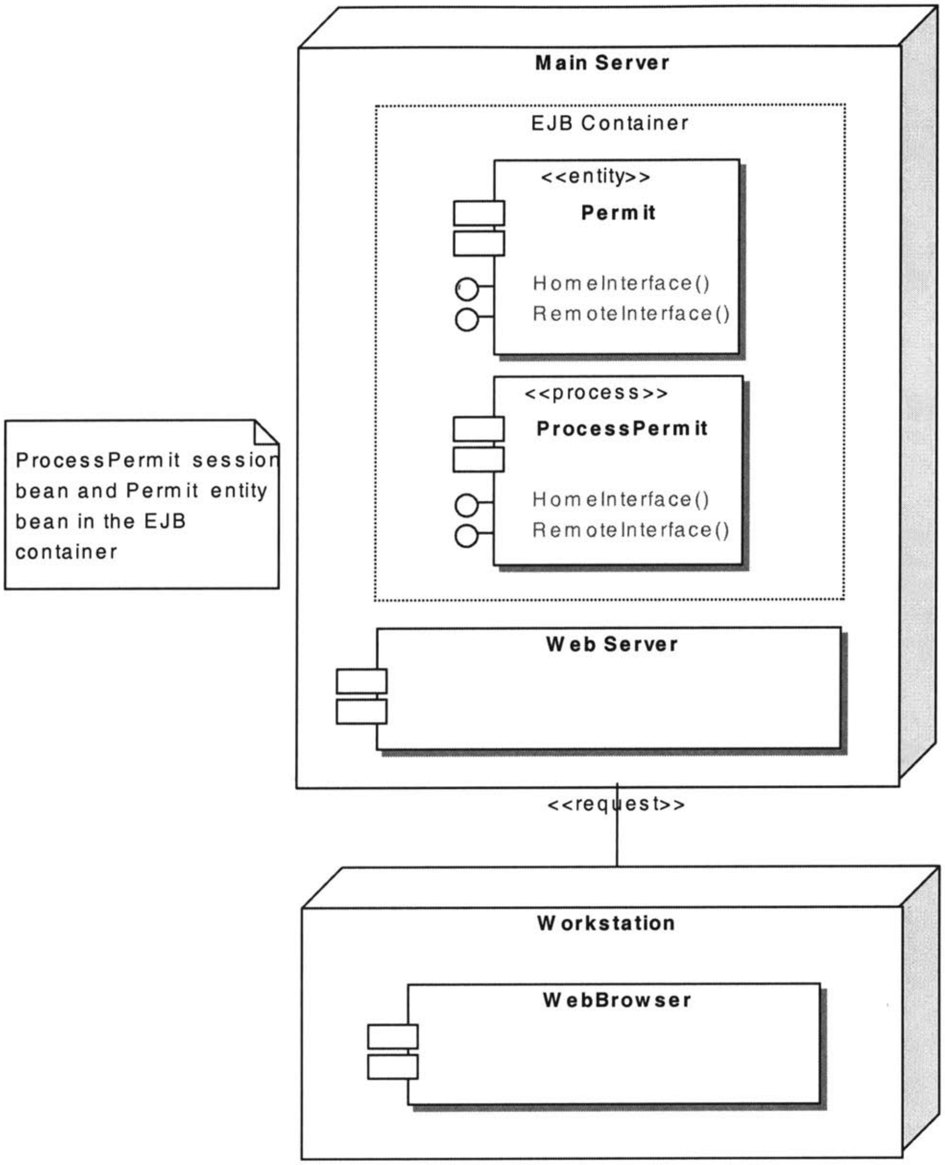


Figure 5-2 Example to Show Enterprise Java Beans Concept

The following example shows how a Permit entity bean might be accessed from a ProcessPermit session bean. Here the homepermit interface is the PermitHome type and the remote interface is the Permit type.

```
// Obtain a reference that implements the home interface
InitialContext ctx = new InitialContext();
```

```
Object objref = ctx.lookup("permit");
PermitHome homepermit = (PermitHome) PortableRemoteObject.narrow(objref,
                                                            PermitHome.class);
// Use the home interface to create a new instance of the Home bean.
Permit thePermit = homepermit.create(perid, percomname, perworktype,
                                              perworkarealength, perapplicationfee);
// using a business method on the Permit.
thePermit.getperapplicationfee()
```

The remote interface defines the business methods of a bean that are specific to the business concept it represents. Below is the definition of a remote interface for a Permit bean.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Permit extends EJBObject {

          public String getperid() throws RemoteException;
          public String getpercomname() throws RemoteException;
          public String getperworktype() throws RemoteException;
          public int getperworkarealength() throws RemoteException;
             public int getperapplicationfee() throws RemoteException;
}
```

5.1.2.2 Business Methods

Business methods can also represent tasks that a bean performs. Both entity beans and session beans can have task-oriented methods, but session beans do not represent data like entity beans.

The following shows the remote interface for a ProcessPermit bean, which is a session bean.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import javax.ejb.DuplicateKeyException;
import javax.ejb.CreateException;

public interface ProcessPermit extends EJBObject {

   public Permit processPermit(String perid, String percomname, String
      perworktype, String retperid, int perworkarealength, int basicfee, int
      additionalfee[], int from[], int to[])        throws RemoteException,
      DuplicateKeyException, CreateException;

    public  Permit getRecord(String perid)  throws RemoteException;
}
```

The business method processPermit defined in the ProcessPermit remote interface represents processes instead of a simple accessor. The ProcessPermit bean acts as if it were an administrator who processes a permit and uses Permit bean to store the necessary information. But we do not need to put the information of PrecessPermit into our database because we only need that "Permit Processor" to perform the administrative task for us. This kind of bean is usually a session bean.

As we already knew from the previous discussion, there are two kinds of enterprise beans: entity beans that represent data in a database, and session beans that represent processes or act as agents performing tasks.

5.1.2.3 Entity Beans

Corresponding to every remote interface, there is an implementation class called the bean class. The bean class is the key element of the bean that actually implements the business methods defined in the remote interface. The Permit bean class is an entity bean that implements the remote interface.

```
//Container managed entity bean

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;

public class PermitBean implements EntityBean {

    public String perid;
    public String percomname;
    public String perworktype;
    public int perworkarealength;
    public int perapplicationfee;

    public String getperid() {
      return this.perid;
    }

    public String getpercomname() {
      return this.percomname;
    }

    public String getperworktype() {
```

```
 return this.perworktype;
}

public int getperworkarealength() {
 return this.perworkarealength;
}

public int getperapplicationfee() {
 return this.perapplicationfee;
}

public String ejbCreate(String perid, String percomname, String perworktype,
        int perworkarealength, int perapplicationfee)
          throws CreateException {
 this.perid = perid;
 this.percomname = percomname;
 this.perworktype = perworktype;
 this.perworkarealength = perworkarealength;
 this.perapplicationfee = perapplicationfee;
 return null;
}

public void ejbRemove() throws RemoteException { }
public void setEntityContext(javax.ejb.EntityContext ctx){ }
public void ejbActivate() { }
public void ejbPassivate() { }
public void ejbPostCreate(String perid, String percomname, String
        perworktype, int perworkarealength, int perapplicationfee) {}
public void ejbLoad() { }
public void ejbStore() { }
public void unsetEntityContext(){ }
```

```
}
```

The application can use the remote interface to the Permit bean to access permit data without any need to write database access logic. Entity beans implement the javax.ejb.EntityBean type that defines a set of notification methods that the bean uses to interact with its container.

5.1.2.4 Session Beans

The ProcessPermit bean is a session bean. Session beans represent a set of processes or tasks, which are performed on behalf of the client application. Session beans may use other beans to perform a task or access the database directly. The ProcessPermit session bean uses Permit entity bean to access the database directly.

```
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import javax.ejb.DuplicateKeyException;
import javax.ejb.CreateException;

public class ProcessPermitBean implements SessionBean {
 PermitHome homepermit;

 public Permit processPermit(String perid, String percomname, String
      perworktype, String retperid, int perworkarealength, int basicfee, int
      additionalfee[], int from[], int to[])
              throws DuplicateKeyException, CreateException {
  Permit thePermit = null;
  int perapplicationfee = basicfee;

  // Process additional fees (for water)
            if ((perworktype.equals("Sewer and water")) ||
                    (perworktype.equals("Water only")))
  {
    // Add the additional fee according to the length
    for (int i=0; i<3; i++)
    {
        if ((perworkarealength>=from[i]) && (perworkarealength<to[i]))
            perapplicationfee+=additionalfee[i];
    }

    // If out of range, assign maximum fee.
    if (perworkarealength>=to[2]) perapplicationfee+=additionalfee[2];
  }

  try {
   InitialContext ctx = new InitialContext();
   Object objref = ctx.lookup("permit");
      homepermit = (PermitHome)PortableRemoteObject.narrow(objref,
               PermitHome.class);
  } catch (Exception NamingException) {
   NamingException.printStackTrace();
  }

//Store data in entity Bean
  try {
   thePermit = homepermit.create(perid, percomname, perworktype,
                        perworkarealength, perapplicationfee);
  } catch (java.rmi.RemoteException e) {
   String message = e.getMessage();
   e.printStackTrace();
  }
  return thePermit;
 }

 public Permit getRecord(String perid) {
  Permit record=null;
//Use primary key to retrieve data from entity Bean
  try {
   record = homepermit.findByPrimaryKey(perid);
  } catch (java.rmi.RemoteException e) {
   String message = e.getMessage();
  } catch (javax.ejb.FinderException e) {
   e.printStackTrace();
  }
  return record;
 }
 public void ejbCreate() { }
 public void setSessionContext(SessionContext context) { }
 public void ejbRemove() { }
 public void ejbActivate() { }
 public void ejbPassivate() { }
 public void ejbLoad() { }
 public void ejbStore() { }
}
```

EJB does not require that the bean class implement the remote or home interfaces but the bean class does provide implementations for all the business methods defined in the remote interface.

5.1.2.5 Methods in the Home Interface

Besides a remote interface, all beans have a home interface. The home interface provides life cycle methods for creating, destroying, and locating beans that are not specific to a single bean instance. The definition of the home interface for the Permit bean is as follows:

```
        import javax.ejb.EJBHome;
        import javax.ejb.CreateException;
        import javax.ejb.FinderException;
        import java.rmi.RemoteException;
        public interface PermitHome extends EJBHome {
          public Permit create(String perid, String percomname, String perworktype, int
                                        perworkarealength, int perapplicationfee)
                       throws CreateException, RemoteException;
          public Permit findByPrimaryKey(String perid)
                   throws FinderException, RemoteException;
      }
```

The create() method is used to create a new entity, which results in a new record in the database. A home may have many create() methods. The number and data type of the arguments of each create() are left up to the bean developer, but the return type must be the remote interface data type. The findByPrimaryKey() method is used to locate specific instance of the Permit bean and we may define as many find methods as necessary.

5.1.2.6 Application of the Remote and Home Interfaces

The remote and home interfaces are used by applications to access enterprise beans at runtime. The home interface allows the application to create or locate the bean, while the remote interface allows the application to invoke a bean's business methods. The following code shows the concept:

```
 ProcessPermitHome homeprocesspermit;
 ProcessPermit theProcessPermit;
 Permit thePermit, retrecord;
 InitialContext ctx = new InitialContext();
 Object objref = ctx.lookup("processpermit");
 homeprocesspermit =(ProcessPermitHome)PortableRemoteObject.narrow
                                  (objref, ProcessPermitHome.class);
//Create remote interface
 theProcessPermit = homeprocesspermit.create();
//Call session bean to calculate the application fee
 thePermit = theProcessPermit.processPermit (perid, percomname,
   perworktype,  retperid, perworkarealength, basicfee, additionalfee, from, to);
//Retrieve record from the database
 retrecord = theProcessPermit.getRecord(retperid);
//Display data returned by session Bean
 out.println("<H1>Permit You Want to Retrieve</H1>");
 out.println("<P>Permit Number: " + retrecord.getperid() + "<P>");
 out.println("<P>Contractor/Utility Company: " + retrecord.getpercomname()
                  + "<P>");
 out.println("<P>Type of Work: " + retrecord.getperworktype() + "<P>");
 out.println("<P>Length of Working Area (Feet): " +
                    retrecord.getperworkarealength() + "<P>");
 out.println("<P>Application Fee ($): " + retrecord.getperapplicationfee()
                    + "<P>");
```

5.2 Enterprise JavaBeans Deployment

In previous section, by using the permit application as an example, we have already described the components of Enterprise JavaBeans. In this section, we will demonstrate how to build, deploy and run an Enterprise JavaBeans application using the source files described in previous section. J2EE SDK only supports version 6.1 of Redhat Linux but it supports Windows 2000, so in this section we limit our discussion to J2EE SDK under Windows 2000 environment. Because Cloudscape DBMS is included in the J2EE SDK, we also use Cloudscape for our system in this section.

5.2.1 Preparation Work

Before we actually start deploying the permit application, we need to do some preparation work as shown below:

5.2.1.1 Building the Permit Application

To build the permit application code we need to install ant as well as J2EE SDK. Ant is a portable Java-based make tool that is hosted by the Jakarta project at the Apache Software Foundation. Ant can be downloaded from: http://jakarta.apache.org/builds/jakarta-ant/release/v1.2/bin.

By default ant uses a configuration file build.xml as the makefile. In order to build the permit application, we need to edit the relevant lines in the build.xml file as shown below:

```
<property name="j2ee-home" value="C:\qcx\prog\j2sdkee1.3" />

 <target name="ejb" depends="init, permit" >
```

```
        <chmod perm="go+r" type="both">
            <fileset dir=".." />
        </chmod>
    </target>

    <target name="permit" depends="init">
        <mkdir dir="${build}/${ejb}/permit" />
        <javac srcdir="${ejb}/permit" destdir="${build}/${ejb}/permit"
                        classpath="${j2eepath}" />
    </target>
```

The above editing is based on the assumption that J2EE SDK installation directory is in C:\qcx\prog\j2sdkee1.3 and all the java source files for permit application are in permit sub-directory under the ejb directory.

After editing the build.xml file, simply execute “ant permit” command to build the permit application.

5.2.1.2 Checking the Environment Variables

The environment variables should be set to the values described in the following table.

Table 5-1 Environment Variables Checklist

ENVIRONMENT VARIABLE	VALUE
JAVA_HOME	Location of the J2SE SDK installation.
J2EE_HOME	Location of the J2EE SDK installation.
ANT_HOME	Location of the ant installation.
PATH	Include the bin directory of the J2EE SDK and ant installation.

5.2.1.3 Starting the Platform and Tools

Before we can assemble, deploy and run the permit application, we need to start the J2EE server, the database server and the deploy tool.

To launch the J2EE server, type this command:

```
j2ee -verbose
```

To launch the database server, type this command:

```
cloudscape –start
```

To launch the deploy tool, type this command:

```
deploytool
```

5.2.2 Assembling the Permit Application

The permit application contains three J2EE components: an enterprise session bean, an enterprise entity bean and a web client. In order to assemble all these components together, we need to create a new J2EE application, packaging the enterprise beans and packaging the web client.

5.2.2.1 Creating the J2EE Application

Once we have launched the deploy tool, we should see a GUI tool as shown in Figure 5-3.

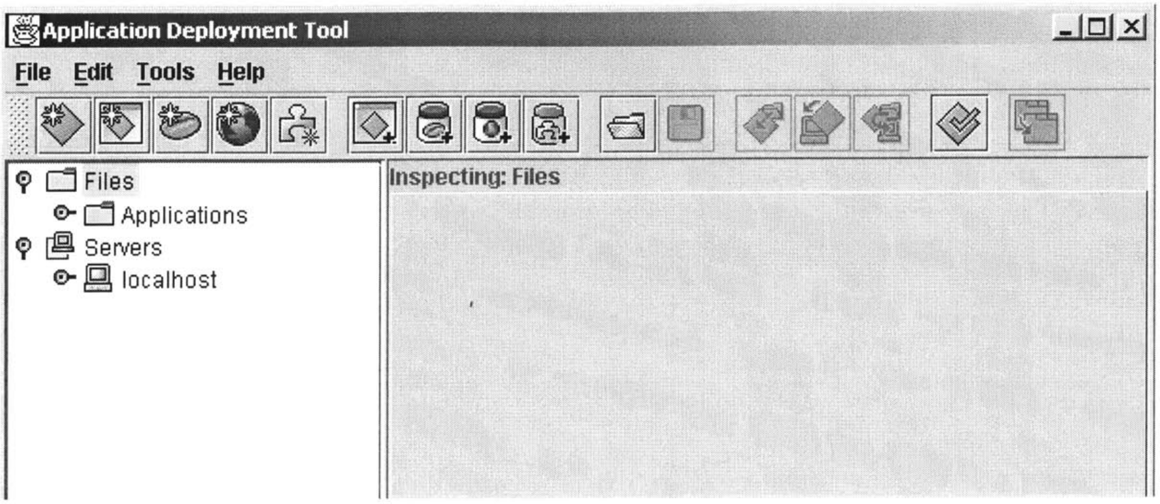


Figure 5-3 Deploy Tool

1. In the GUI deploy tool, select New Application from the File menu.
2. Locate the directory where you want to place the application EAR file. In our case, the directory is ../src/ejb/permit.
3. In the File name field enter PermitApp.ear.
4. Click New Application and then OK.

5.2.2.2 Packaging the Web Client

To package a web component, we run the New Web Component Wizard of the deploytool. During this process, the wizard puts the client files (PermitServlet.class servlet files and permit.html file) into a WAR file and then adds the WAR file to the application's PermitApp.ear file.

To start the New Web Component Wizard, select New Web Component from File menu. The wizard displays the following dialog boxes.

1.Introduction Dialog Box:

- Read and Click Next

2.WAR File General Properties Dialog Box

- In the combo box labeled Create New WAR File in Application, select PermitApp.
- In the WAR Display Name field, enter PermitWAR and Click Add.
- Navigate to ../build/ejb/permit directory and add PermitServlet.class
- Navigate to ../src/ejb/permit directory and add permit.html
- Click Finish

3.Component Type Dialog Box

- Select the servlet radio button and Click Next

4.Component General Properties Dialog Box

- Make PermitServlet the servlet class
- Make the display name PermitServlet

5. Component Aliases Dialog Box

- Specify PermitAlias and Click Finish

5.2.2.3 Packaging the Enterprise Beans

In this section we will run the New Enterprise Bean Wizard of the deploytool to perform these tasks:

- Creating and packaging the ProcessPermit session bean's deployment descriptor and the bean's classes in an EJB JAR file.
- Creating and packaging the Permit entity bean's deployment descriptor and the bean's classes in that same EJB JAR file just created.
- Insert the EJB JAR file into the application's PermitApp.ear file.

To package the ProcessPermit session bean into an EJB JAR file, select New Enterprise Bean from File menu. The wizard displays the following dialog boxes.

1.Introduction Dialog Box

- Read and Click Next.

2.EJB JAR Dialog Box

- In the combo box labeled Enterprise Bean Will Go In, select PermitApp.
- In the JAR Display Name field enter PermitJAR.
- Click the Add button next to the Contents text area.

- Select the following classes from the beans directory and click Add: ProcessPermit.class, ProcessPermitBean.class, and ProcessPermitHome.class.

3.General Dialog Box

- Under Bean Type, select the Session and Stateless radio button.
- In the Enterprise Bean Class combo box, select ProcessPermitBean.
- In the Home Interface combo box, select ProcessPermitHome.
- In the Remote Interface combo box, select ProcessPermit.
- In the Enterprise Bean Name field, enter ProcessPermitBean.

4.Dialog Boxes for Environment Entries, Security etc.

- Click Next continuously until come to the Transaction Management dialog box.

5. Transaction Management Dialog Box

- Select Container-managed transactions.
- In the list below make processPermit and getRecord required and Click Next then Finish.

To package the Permit entity bean into the same EJB JAR file, select New Enterprise Bean from File menu. The wizard displays the following dialog boxes.

1.Introduction Dialog Box

- Read and Click Next.

2.EJB JAR Dialog Box

- In the combo box labeled Enterprise Bean Will Go In, select PermitJAR. This will add the new bean to the existing JAR file instead of creating a new JAR file.

- In the JAR Display Name field enter PermitJAR.
- Click the Add button next to the Contents text area.
- Select the following classes from the beans directory and click Add: Permit.class, PermitBean.class, and PermitHome.class.

3.General Dialog Box

- Under Bean Type, select the Entity radio button.
- In the Enterprise Bean Class combo box, select PermitBean.
- In the Home Interface combo box, select PermitHome.
- In the Remote Interface combo box, select Permit.
- In the Enterprise Bean Name field, enter PermitBean.

4.Entity Setting Dialog Box

- Select Container managed persistence.
- In the list below check perid, percomname, perworktype, perworkarealength, perapplicationfee. The primary key class is java.lang.String, and the primary key field is perid. Click Next.

4.Dialog Boxes for Environment Entries, Security etc.

- Click Next continuously until come to the Transaction Management dialog box.

5. Transaction Management Dialog Box

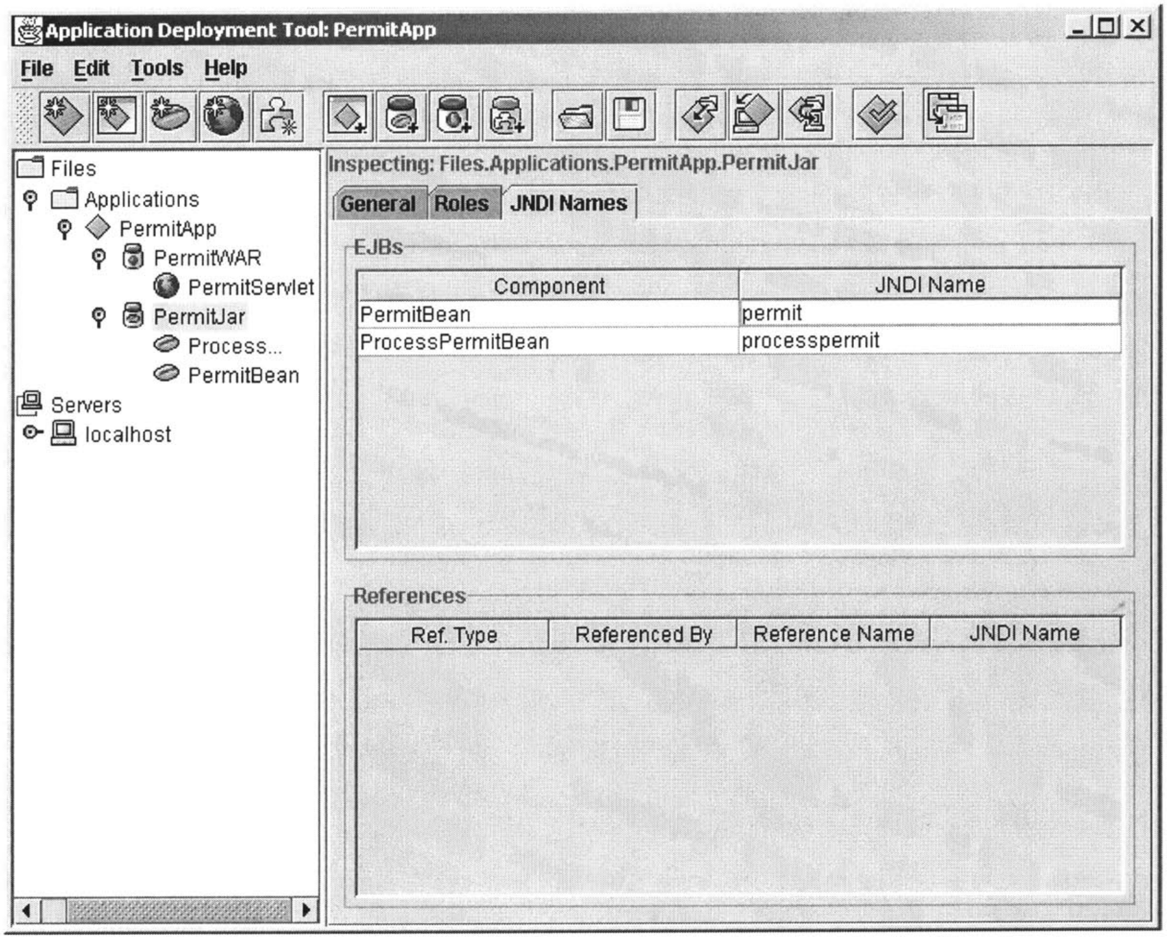
- Select Container-managed transactions.
- In the list below make create, findByPrimaryKey, getperid, getpercomname, getperworktype, getperworkarealength and gerperapplicationfee required and Click Next then Finish.

After all these have been done, we need to set JNDI names. In the Local Applications window, select PermitApp, and in the Inspection window, select JNDI names tab, give PermitBean the JNDI name of permit and ProcessPermitBean the JNDI name of processpermit.

Before we can deploy the J2EE permit application, we need to specify the deployment setting for the Permit entity bean and generate the SQL. In the local application window, select PermitBean, and in inspecting window, select Entity tab and click the Deployment Setting button. And in the Deployment Setting window do the following operation:

- Type jdbc/Cloudscape for the Database JNDI name.
- Check the Create table on deploy and Delete table on undeploy boxes.
- Click Generate SQL.

Now we should have all the necessary components ready for deployment. The application deployment tool now should look like Figure 5-4.



Application Deployment Tool: PermitApp

Inspecting: Files.Applications.PermitApp.PermitJar

EJBs

Component	JNDI Name
PermitBean	permit
ProcessPermitBean	processpermit

References

Ref. Type	Referenced By	Reference Name	JNDI Name

Figure 5-4 Components for Permit Application Ready for Deployment

5.2.3 Deploy the J2EE Application

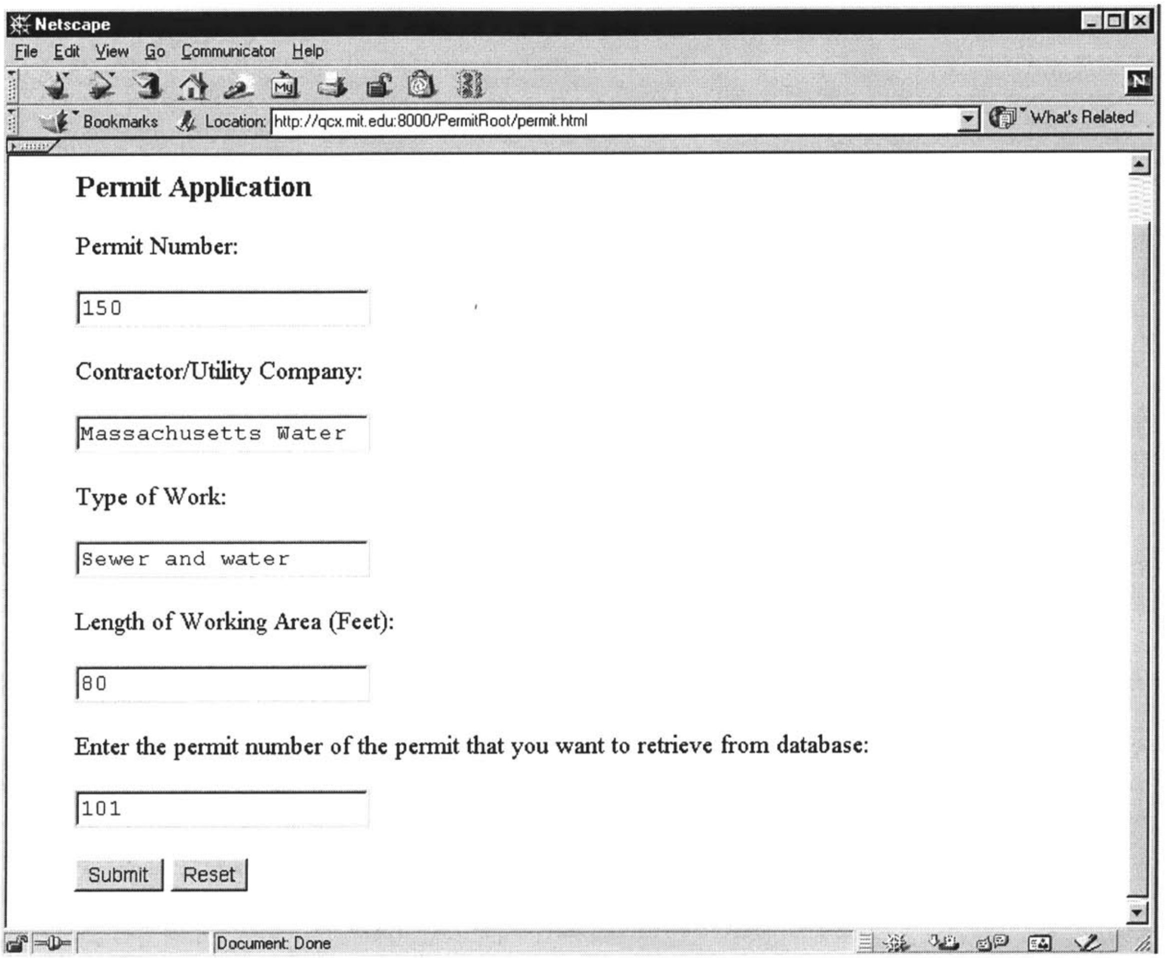
Now it is possible to deploy the Permit application.

- Select Deploy Application from the Tools menu.
- Set the Target Server as localhost.
- Click Next and make sure the JNDI names show processpermit for ProcessPermitBean and permit for PermitBean.
- Click Next and make sure the Context Root name shows PermitRoot.
- Click Next then Finish to start the deployment.
- When deployment is completed, click OK.

5.2.4 Run the J2EE Application

To run the web application described in this chapter, go to the URL:
http://J2EEServer:8000/PermitRoot/permit.html

Where J2EEServer should be replaced by the name of the host running the J2EE server. After input the permit application information such as contractor/utility company name, street work type, length of work area etc, the application can calculate the application fee accordingly and stored information in the database. It can also retrieve permit information from database. A simple example is shown in Figure 5.5 and Figure 5.6.


Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://qcx.mit.edu:8000/PermitRoot/permit.html
What's Related

Permit Application

Permit Number:

150

Contractor/Utility Company:

Massachusetts Water

Type of Work:

Sewer and water

Length of Working Area (Feet):

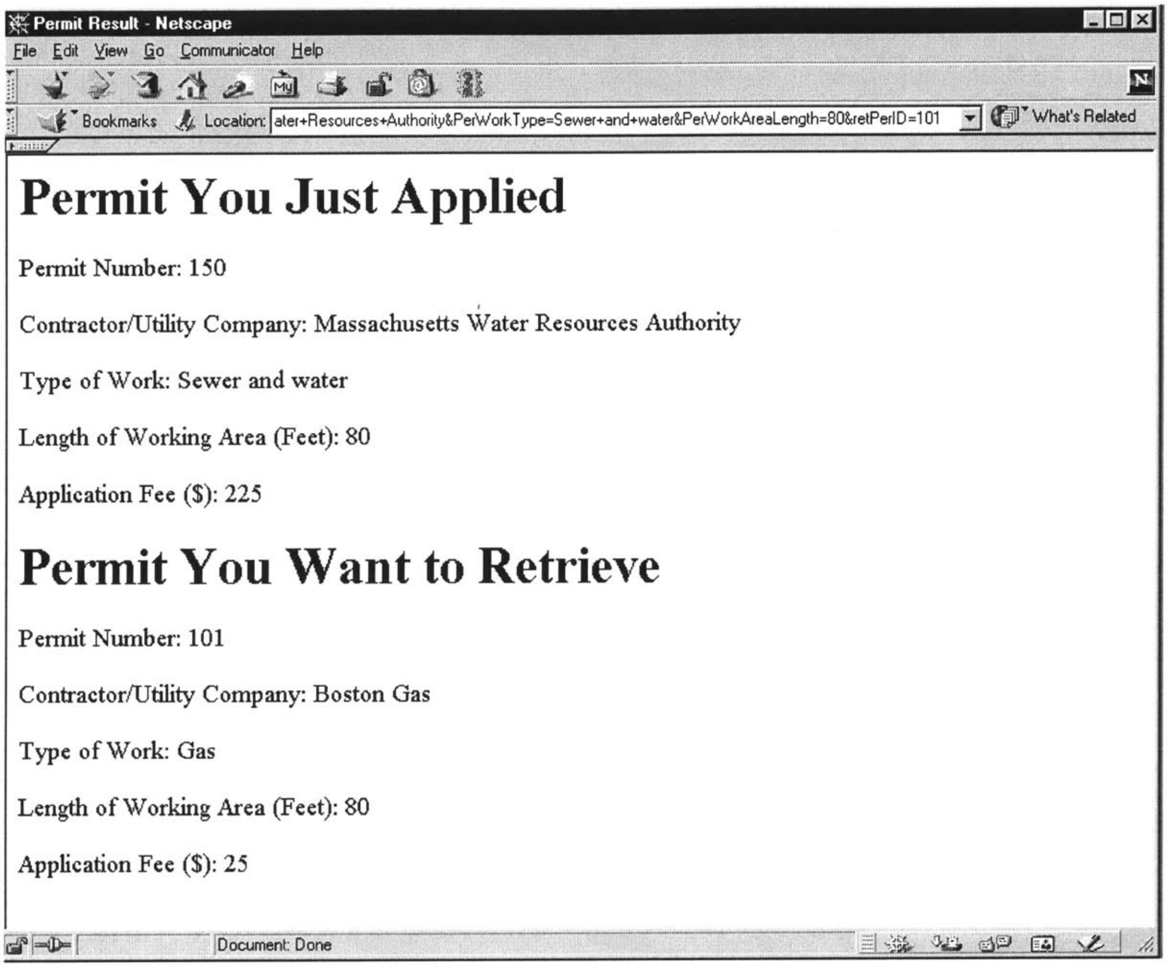
80

Enter the permit number of the permit that you want to retrieve from database:

101

Submit
Reset
Document: Done

Figure 5-5 Permit Application


Permit Result - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: ater+Resources+Authority&PerWorkType=Sewer+and+water&PerWorkAreaLength=80&retPerID=101
What's Related
Document: Done

Permit You Just Applied

Permit Number: 150

Contractor/Utility Company: Massachusetts Water Resources Authority

Type of Work: Sewer and water

Length of Working Area (Feet): 80

Application Fee ($): 225

Permit You Want to Retrieve

Permit Number: 101

Contractor/Utility Company: Boston Gas

Type of Work: Gas

Length of Working Area (Feet): 80

Application Fee ($): 25

Figure 5-6 Retrieve Permit Application from Database

6 Conclusions

6.1 Summary of the Work

This thesis is focused on the enabling technologies for a web-based urban street construction permit system. The main J2EE technologies we have discussed includes:

- Enterprise JavaBeans
- JavaServer Pages
- Servlet
- JDBC API

Besides, other J2EE technologies such as Transaction, JNDI and XML are also discussed where appropriate.

The following development environments to support these technologies are also described in detail:

- Red Hat Linux 7.0
- Java 2 Platform
- Tomcat Server 3.2.1
- Database MySQL 2.1.4
- JDBC Driver 2.0.4 for MySQL

As an example, Arlington permit system was used to demonstrate the design of Entity-Relational model, and the Enterprise JavaBeans application.

6.2 Future Work

As we already described in Chapter 1, the current practice of issuing permit for urban street construction is very tedious and inefficient. To solve the problem, we developed a web-based urban street construction permit system that mainly use JavaServer pages, Servlets, Java bean to implement the system.

As we know, street permit issuing is a typical activity for all cities and towns in the U.S.; the system we developed should find extensive use in other American towns besides Arlington town. However, the scale of the town, the computing platforms in the town's relevant department and other environment are quite different from town to town, and for the permit system to be extensively used in U.S. towns, it is crucial that the permit system should be robust, scalable, and portable. Enterprise JavaBeans approach proves to be an effective way of realizing that purpose.

Future work should focus on the following aspects:

- From our database normalization analysis in Chapter 4, we found that there were several cases in which it is better to separate some attributes from their original table and put them in a new table. Just for the simplicity of the Arlington Permit System, we did not implement these separations. But in order to develop a general permit application system that can be applied to different towns in the U.S., we should separate these tables although it does add some complexity.
- In Chapter 5, a prototype permit application system was developed using the Enterprise JavaBeans approach, but effort is needed to make it a completed permit system. We developed our prototype permit system on Java 2 Platform, Enterprise Edition 1.3 Beta release because it is free, but for a real project application, we need to implement it on an earlier but more stable release of the platform.
- The differences and similarities of the requirements for permit systems for different towns in the U.S. should be fully studied so we can develop several types of business logic to cater for each town's particular needs.

- A typical platform and a reference implementation of the permit system may be recommended for each type of business logic mentioned above.

Bibliography

[Arlington2001a] "Arlington Online", http://www.town.arlington.ma.us/, 2001

[Arlington2001b] Klimke,Wolfgang Andreas, Qi, Changxin and Prasad, Rajesh. "Street Opening Permit System Project Report", MIT, 2001

[Asbury1999] Asbury, Stephen and Weiner, Scott R. "Developing Java Enterprise Applications", John Wiley & Sons, 1999

[Bodoff2001] Bodoff, Stephanie et al. "The J2EE Tutorial", Sun Microsystems, Inc., 2001

[Elmasri1999] Elmasri, Ramez and Navathe, Shamkant B. "Fundamentals of database systems", Third Edition, Benjamin/Cummings Publishing, 1999

[Gilbert1998] Gilbert, Stephen and McCarty, Bill. "Object-Oriented Design in Java", Sams,1998

[Hall2000] Hall, Marty. "Core Servlets and JavaServer Pages", Prentice-Hall Inc., 2000

[Jakarta1999] "The Jakarta Project, Tomcat", http://jakarta.apache.org/tomcat/index.html, 1999

[Matthews2001] Matthews, Mark. "MM MySQL JDBC driver", http://mmmysql.sourceforge.net/, 2001

[MySQL1995] "MySQL Reference Manual", http://www.mysql.com/doc/index.html, 1995

[RedHat2000] "Red Hat Linux 7.0, The Official Red Hat Linux Reference Guide", Red Hat, Inc., 2000

[Roth1998] Roth, Bill. "An Introduction to Enterprise JavaBeans Technology", http://developer.java.sun.com/developer/technicalArticles/ebeans/IntroEJB/, 1998

[SUN1999] "Java 2 Platform, Enterprise Edition, Overview", http://java.sun.com/j2ee/overview2.html, 1999

[SUN2000a] "Java 2 Platform, Standard Edition, Version 1.3", http://java.sun.com/j2se/1.3/, 2000

[SUN2000b] "Java 2 Platform, Standard Edition for Linux", http://java.sun.com/j2se/1.3/download-linux.html, 2000

[SUN2001a] "Java 2 Platform, Enterprise Edition Blueprints", http://java.sun.com/j2ee/blueprints/, 2001

[SUN2001b] "Java 2 SDK, Enterprise Edition 1.3 Beta release", http://developer.java.sun.com/developer/earlyAccess/j2ee/, 2001

[Zakon2001] Zakon, Robert H. "Hobbes' Internet Timeline v5.3", http://www.zakon.org/robert/internet/timeline/#Growth, 2001