# Web Application Development
# Using Open Source and Java Technologies

by
Wolfgang Andreas Klimke
Diplom-Bauingenieur
University of Stuttgart, Germany, 1998

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 of Civil and Environmental Engineering
Thesis Supervisor

Accepted by:

Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies

# Web Application Development
# Using Open Source and Java Technologies

by
Wolfgang Andreas Klimke

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

With the growing popularity of Open Source and Java, software engineers have embraced the idea of a no-cost development platform for Web applications. This thesis introduces the concepts of Open Source and gives an overview on current Open Source products with particular focus on Web application enabling technologies.
Covered are the technical aspects of Web application development with today's leading Web application development tools, such as Tomcat (the reference implementation of the Java Servlet API) and the Concurrent Versions System CVS for source code control.
A case study of a real-world application, the development of a Web-based permit system for urban road construction, is presented to show that Open Source development can meet the expectations it has created.
Java is the dominant Web programming language today, and is used extensively by the Open Source community. The author takes a closer look at ASP and C#, representing the core technologies of Microsoft's Web application development platform, as Java's most important competitor.

Thesis Supervisor: George Kocur
Title: Senior Lecturer of Civil and Environmental Engineering

## Acknowledgements

I would like to thank those who made it possible for me to attend MIT, for their great support and inspiration:

*Mr. Ivan Karlin of Holderbank Engineering, Canada*
*Prof. Dr.-Ing. Stefan Holzer of the University of Stuttgart, Germany*

MIT is an amazing place to be, and I am very grateful for the fantastic year I have spent here. My special thanks go to:

*Dr. George Kocur of the Massachusetts Institute of Technology*

The case study presented in Chapter 4 was conducted in close collaboration with the Department of Public Works of the Town of Arlington. Special thanks to:

*Mr. Ron Santosuosso of the Engineering Division*

Special thanks to my family and my girlfriend Grace for their ongoing support and love!

# Table of Contents

## List of Figures



## List of Tables

# 1 Introduction

Web Technologies, particularly the Internet, have become an important part of the business world over the past few years. People have become accustomed to searching the Internet for data, sending e-mails, or making simple purchases electronically. Web browsers, such as Microsoft's Internet Explorer, or Netscape's Navigator, played an important part in the rapid development of the Internet. Designed to present a simple and easy-to-operate front-end interface to the user, Web browsers are today's standard way of accessing the vast amount of information available through the Internet.

More recently, with more and more people being connected and the steadily increasing connection speed to the Internet, a new wave of more sophisticated web applications emerged. In order to make these applications possible, the browser's original functionality of "browsing", i.e. navigating through the contents of web sites via hyperlinks, needed to be extended significantly. Scripting languages, such as VBScript or JavaScript, Java-enabled browsers, DHTML, or XML are some of the technologies that are part of today's browsers to provide the means for the development of real-world applications such as on-line bookstores, on-line travel agencies, or banking services, which are some of the most well-known examples for modern Web usage. The technologies mentioned above served to enhance the mostly "static" contents (e.g. text or images) of Web pages with "dynamic" elements, such as event handling for error checking of forms prior to submittal, or performing of calculations. But not only browsers needed to be improved. More importantly, Web servers needed to be able to respond to client requests in a more flexible way than presenting the same content to all users. Server-side programming is the key technology that allows developers to make Web applications fit for a specific purpose, so incoming requests can be processed by the server on the fly (e.g. presenting filtered information of a database according to search criteria provided by the client).

Server-side programming is today's way of choice to develop Web applications. A Web server can be completely controlled by the application provider, eliminating uncertainties regarding the capabilities of the client's Web browser. With server-side programming, Web pages are prepared by the application considering the client's request. This preparation happens on the server, so that the information submitted to the client can be in HTML format, the Internet's basic, standardized language of communication.

Several server-side technologies exist today. The first solution to bringing dynamic data to the Web was CGI, the Common Gateway Interface. CGI provided a simple way to create a Web application that accepts user input, queries a database, and returns responses to the client. Then, both Microsoft and Netscape developed proprietary, server-specific API's to answer inefficiencies of CGI (especially its poor scalability). However, these API's were limited to a particular platform. Furthermore, these API's reduced the stability of the server due to the fact that their programs had to run within the same process as the Web server. The next step in the continuing development of server-side technologies was Microsoft's Active Server Pages (ASP), and Sun's Java Servlets and Java Server Pages (JSP). Implementations for JSP and Servlets are available on many platforms, and they provide a good way of separating code and HTML to increase maintainability. JSP and Servlets have become today's most popular way of server-side programming for enterprise applications, while ASP dominates among smaller sites.

An effort to improve Microsoft's position in the server-side application development market was made with the introduction of the .NET platform in 2000. The goal of .NET was to regain some of the market that was lost to Sun Microsystems due to the superior concept of Servlets/JSP over ASP. Sun, on the other hand, recently announced Sun ONE to compete with the .NET strategy, showing that server-side programming is still evolving and improving at a rapid pace.

Since the server-side application development market is still evolving, developers are confronted with the difficulty of having to work with constantly changing techniques, and having to make the right choice of which development platform to use.

This thesis illuminates the concept of an Open Source, "free of charge" development platform and server implementation, which has enjoyed an increasing popularity. The advantages are numerous: The latest Open Source operating system, Linux, features high reliability, stability and a transparent technology at no cost, making it a very appealing alternative to Microsoft Windows, SunOS or other commercial offerings. Results of a current market share analysis among SSL Site operating systems in the United States are shown in Figure 1. Linux captures a significant market share.

In other countries, the Linux operating system is even more popular. In Germany, for example, Linux had over 30% market share among SSL Site operating systems according to surveys conducted in February 2001.

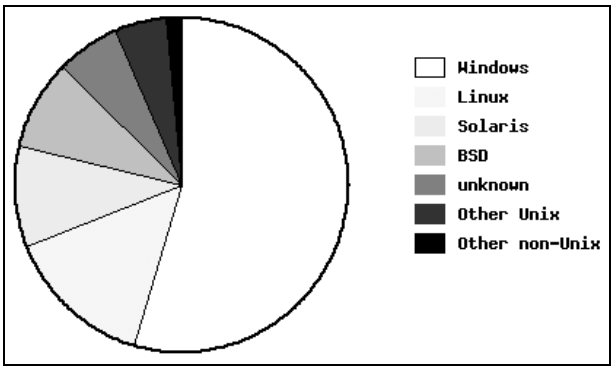


**Figure 1: SSL Site Operating Systems in the US, February 2001**

Source: http://www.netcraft.com/survey/

Java, although not an Open Source product, has become the dominant Web programming language due to its complete class library and easy to use syntax, and is also available free of charge. Open Source databases have matured and provide enough stability to be used in quite large applications as a low-budged alternative to Oracle or SQL Server 2000. Last but not least, the Open Source Web server Apache is actually dominating the market, as shown in recent studies (see Table 1).

| Server Type | Total # of Sites | Market Share in % |
|---|---|---|
| Apache | 16,871,744 | 59.99% |
| Microsoft-IIS | 5,522,069 | 19.63% |
| Netscape-Enterprise | 1,751,123 | 6.23% |
| WebLogic | 1,039,605 | 3.70% |
| Zeus | 801,215 | 2.85% |
| Rapidsite | 380,217 | 1.35% |
| Thttpd | 367,724 | 1.31% |
| tigershark | 166,465 | 0.59% |
| AOLserver | 153,296 | 0.55% |
| WebSitePro | 114,655 | 0.41% |

**Table 1: Market Share for Top Servers, February 2001**

Survey Results (over 28 Million responses). Source: http://www.netcraft.com/survey/

The major downsides of Open Source are twofold. First, accurate documentation is often more difficult to find, and second, Open Source products are usually harder to configure than their

commercial counterparts. This document addresses these issues for today's most popular Web application Open Source products by taking a close look at what is available, and giving the reader a starting point by providing a technical introduction and many selected references.

# 2 Current Open Source Standards – A Brief Overview

## 2.1 What is Open Source?

Before diving into the different Open Source technologies, it is helpful to look at where Open Source came from, and the motivation behind it.

When developing applications, it is important to understand both the rights and obligations that come with using Open Source software, since this can affect the newly developed product itself. While Open Source implies that the source code is available and modifiable, the use of the program is free, and copies may be made, it often also means that the same rules have to apply to the program that utilizes Open Source, depending on the type of license that the Open Source software is provided under.

It is also worth mentioning that free software, freeware, public domain, and Open Source don't mean the same thing, although all of these products can be obtained free of charge. Recommend references are given below for readers interested in a more detailed discussion of these topics.

The term "Free Software" is based on the idea of making software accessible to anyone for free, including the right for everyone to make improvements to it. This implies that the source code is provided with the software product. To protect the rights of the authors, a formal approach through licenses has been introduced through Richard Stallman, a former member of the MIT Artificial Intelligence Lab, with the founding of the Free Software Foundation (FSF) in 1984. Stallman's goal was to develop a UNIX-compatible operating system called GNU (= GNU's not UNIX) which should give users a portable, no-cost alternative to the various UNIX systems that were specific to each hardware vendor. He planned to achieve this by gradually replacing the proprietary versions of the UNIX system software components with his and other voluntary

contributor's developments. To protect the author's copyright, as well as to guarantee the free use of derivatives of the GNU work, the FSF developed the GNU General Public License (GPL).

Some products available under the GNU Public License, such as EMACS or the GNU C Compiler became widely used, but only the introduction of Linux under the GPL pushed the idea of free software to a new level of interest. With Linux's growing market share, the business world took notice of the free software movement. The restrictive GNU public license with its idealistic view, however, led to the development of other public licenses that seemed more appropriate for business purposes, while taking advantage of the ideas of free software.

The introduction of these other licenses, such as the BSD, MIT X, Mozilla, or Artistic licenses, increased the confusion about the meaning of free software and Open Source. Therefore, community members developed a specification called "The Open Source Definition" in 1997. This specification establishes guidelines for software licenses to be considered "Open Source". The definition is available at http://www.opensource.org/osd.html.

### Selected References


- Donald K. Rosenberg, M&T Books; ***Open Source: The Unauthorized White Pages, Chapter 1 and 3***; 2000;

- Bruce Perens, O'Reilly & Associates, Inc.; ***Open Sources: Voices from the Open Source Revolution: Essay 11: The Open Source Definition***; 1999


## 2.2 Linux Overview

### 2.2.1 Introduction to Linux

Linus Torvalds created the Linux operating system as a personal project in 1991 (in Finland), out of the desire to learn and understand the 386 processor and Unix-based operating systems. It was released free-of-charge to the public for everyone to make improvements under the terms of the GNU General Public License.

Since then, Linux has grown into a major player in the operating system market, thanks to the contribution of hundreds of developers all over the world and the coordinating efforts of Linus Torvalds. It has been ported to run on a variety of architectures including Compaq's Alpha, Sun's SPARC, and Motorola's PowerPC chips. The term "Linux" technically only refers to the kernel (the core of the operating system).

An important reason why Linux has become so popular today is the ease of availability through the so-called “distributions”. Distributions bundle the Linux operating system together with useful applications developed by independent groups, and usually have an installation program. Many companies offer Linux distributions today; a list of major distributions is given in the next section.

With the combined efforts of companies as well as individuals, Linux has evolved into a modern operating system that incorporates protected memory, multitasking, fast TCP/IP networking, shared libraries and multi-user capabilities.

### 2.2.2 Comparison of the Distributions

The following table gives an overview over some of the most popular Linux distributions that are currently available. This table should help in finding the Linux distribution that is most applicable for the desired tasks.

| Distribution** | Version | Description |
|---|---|---|
|  Red Hat | 7.0 | Red Hat Linux is one of the oldest and most popular of today's Linux distributions. Red Hat originated the RPM format used by many other distributions. Most RPM packages that are available on the Internet were compiled on Red Hat systems, and therefore install and run fine on Red Hat. |
|  Yellow Dog | 1.2.1 | Linux distribution for PowerPCs (Mac). Includes many software packages. Includes automated update program for installation. |
|  Elfstone | Beta | Elfstone Software is a newer player in the Linux market. Elfstone claims to have a particularly stable Linux distribution. It was designed primarily for programmers, engineers, and network administrators, and contains only software to operate a network. Based on RPM (upgradeable). |
|  Stampede | 0.9 | The Stampede GNU/Linux Foundation is a non-profit organization. Stampede is still under development. Developers contribute on a voluntary basis. |
|  KSI | 2.0 | Ukrainian/Russian Linux distribution. |

| Distribution** | Version | Description |
|---|---|---|
|  Libranet | 1.9.0 | The Libranet distribution is built on top of Debian. This distribution addresses primarily new users with simple installation, automatic configuration and convenient selection of software packages. |
|  Debian | 2.2r2 | Unlike most other current Linux installations, the Debian installation is text-based, and is therefore not recommended for beginners. Many platforms are supported: x86 Intel machines, Alpha, ARM, Motorola 68K, PowerPC and SPARC. |
|  Storm | Hail 2.06 | Linux distribution based on Debian 2.2r2. Similar to Libranet, Storm Linux 2000 offers an easy installation process. Storm comes with some additional commercial software (Star Office, Acrobat Reader) |
|  Mandrake | 7.2 | Pre-configured graphical Linux operating system with emphasis on the ease of installation. Claims full compatibility with Red Hat Linux. |
|  OpenLinux | 2.3 & 2.4 | Caldera, Inc. offers two Linux distributions, one for servers and one for desktops. The server version eServer 2.3 includes 10 major server products, and easy web-based remote management. The desktop version eDesktop 2.4 is specifically targeted for Internet users. Technical support is available to registered users. |
|  Phat | 3.3 | Phat Linux was the first Linux distribution that could be installed on a Windows partition. |
|  MkLinux | DR3 | Apple Computer's version of the GNU/Linux operating system based on the Mach 3 Open Source project. Versions of MkLinux run on the Intel, PA-RISC, and PowerPC architectures. |
|  SuSE | 7.1 | SuSE Linux AG, headquartered in Nuremberg, Germany, is a well-established company with over 500 employees worldwide. The professional edition of version 7.1 includes easy installation, many development and server products, and 90 days free installation support. It is available for a wide range of hardware architectures including Sparc, Alpha, S/390, RS/6000, and PowerPC, and can therefore be used in heterogeneous networks. |

| Distribution** | Version | Description |
|---|---|---|
| Corel Linux | OS Second Edition | This distribution is based on Debian, and offers an easy 4-step installation process. The desktop is KDE-based. The professional edition includes 30 days installation support and some commercial software products. |
| Best Linux | 2000 R3 | Developed by SOT Finnish Software Engineering, Ltd. Relatively new Linux distribution. Includes manual and games CD. Menu-based installation. Fully upgradeable since RPM-based. Recommended for typical home users. |
| Slackware | 7.0 | Slackware contains an easy to use installation program, online documentation, and a menu-driven package system. A full installation also includes networking utilities, a mail server, a news server, a web server, etc. Slackware Linux is available for x86, Alpha and Sparc systems. |
| ASPLinux | * | ASP Linux is a multinational development company based in Singapore. The distributions claims to be 100% compatible to Red Hat 7.0. |
| LuteLinux | * | LuteLinux.com is based in Vancouver, Canada. In addition to the distribution, LuteLinux offers training and certification services, and management and consulting services. |
| Conectiva | 6.0 | Conectiva, Inc. is based in Brazil, and its Linux distribution is mainly targeted for the South American market. |
| Turbolinux | 6 | Turbolinux, Inc. focuses on integrating the open source Linux system with commercial software offerings from established industry leaders including IBM, Oracle and Computer Associates. Several optimized distributions for enterprise application servers and database servers are available. |
| LinuxPPC | 2000 Q4 | Linux/PPC is the native port of Linux to the PowerPC processor. The Linux PPC project is open source, supported by voluntary contributors. |

**Table 2: Linux Distributions**

* Not explicitly versioned ** Based on: Linux distribution page; www.linux.com/getlinux

### 2.2.3 Selected References

- Tucows, Inc.; ***Linux Console – Distributions***; http://www.linuxberg.com/distribution.html; 2001; Descriptions & ratings of popular Linux distributions.

- Erik Severinghaus; ***Comparison of Linux Distributions***; http://www.linux.com/support/newsitem.phtml?sid=82&aid=6837; February 2000
  The author shares his personal experiences on the following Linux distributions: Caldera, Corel, Debian, Mandrake, Red Hat, and Slackware.

- Rod Smith; ***Linux Distributions Guide***; http://www.rodsbooks.com/distribs/; February 2001
  Description of several popular Linux distributions, including: Caldera OpenLinux 2.4, Corel Linux 1.2, Debian GNU/Linux 2.2, Linux Mandrake 7.2, Linux by Libranet 1.2.2, LinuxPPC 2000, Red Hat Linux 7.0, Storm Linux 2000, SuSE Linux 7.0, and Yellow Dog Linux 1.2.

## 2.3 Open Source Web Technologies

All products introduced in this section are Open Source. They are available at no charge, including source code. However, a responsible use according to each product's license is required.

### 2.3.1 The Apache Web Server

Apache is by far the most popular Web server today, with a market share of about 60% according to the Netcraft Web server survey (see http://www.netcraft.com/Survey/). Apache is maintained by the Apache Software Foundation (http://www.apache.org). Brian Behlendorf and Cliff Skolnick initiated the Apache project in 1995 out of the need for rapid development of new Web server features. The first version of Apache was based on the NCSA HTTPd Web server, and modified through patches (therefore the name: Apache = "a patchy" server). The original code base was eventually replaced in the next revision.

An important year for Apache was 1998 when an agreement with IBM for the continued development of Apache was reached. IBM now includes the Apache code in its WebSphere server product. Today, many voluntary contributions from companies and individuals guarantee for the ongoing success of the Apache Web server.

The following is a list of Apache features for the current version 2.0:

- Serves static and dynamic CGI Web pages. Can interface with many dynamic content generation technologies such as Perl, Java Servlets, or PHP.
- Highly configurable. Apache is composed of many modules that can be optionally added or removed, as well as configured.
- Extensive security features. Several forms of authentication, including SSL encryption are available.
- Portable (implementations are available for many platforms, including Windows, Unix, Linux, and OS/2).

Apache is available for download, but also ships with most Linux distributions.

### 2.3.2 Tomcat Servlet Engine and Web Server

Web applications today are based on the ability of the server to generate flexible, dynamic content. Several technologies are available that address dynamic content generation – Microsoft's Active Server Pages technology (ASP), Java Server Pages and Servlets, or PHP are examples to name but a few.

"Tomcat is the Servlet and Java Server Pages reference implementation" (quote from the Tomcat Web site http://jakarta.apache.org/tomcat/). To explain this statement further: While the Servlet and Java Server Pages technologies are owned by Sun, the development of an engine actually using these technologies has been put under the responsibility of the Jakarta Apache Project. Jakarta is the home of several Open Source projects that all have Java as the common platform. Tomcat is one of these projects.

In addition to being a Servlet and JSP Engine, Tomcat can also be used as a Web server. This is especially useful during development for testing, when performance is not a critical issue. For deployment, however, it is recommended to use Tomcat in conjunction with a more stable and faster Web server, such as Apache.

The latest version of Tomcat can be downloaded at the Internet address given above. On-line documentation covers installation on Windows or Linux, as well as other topics such as Apache-Tomcat Web server integration, or application development under Tomcat. Some of these topics are covered later on in this document.

### 2.3.3 Enhydra Application Server

#### 2.3.3.1 Introduction to Enhydra

Enhydra provides a development environment for creating Web applications, being the Open Source alternative to commercial software products that typically cost several thousand or up to tens of thousands of dollars per CPU (e.g. WebSphere by IBM, iPlanet Application Server by Sun). Enhydra is available for free download at http://enhydra.org, or as a packaged version including hard-copy documentation and support for about $700 (development license) or $1000 (deployment license) from Lutris Technologies, Inc.

A major downside of Enhydra has been its lack of support for the Java 2 Enterprise Edition. This problem is currently being addressed: A beta version of Enhydra Enterprise has recently become available (end of March 2001). Enhydra Enterprise will fully support J2EE.

### 2.3.3.2 The Enhydra Architecture

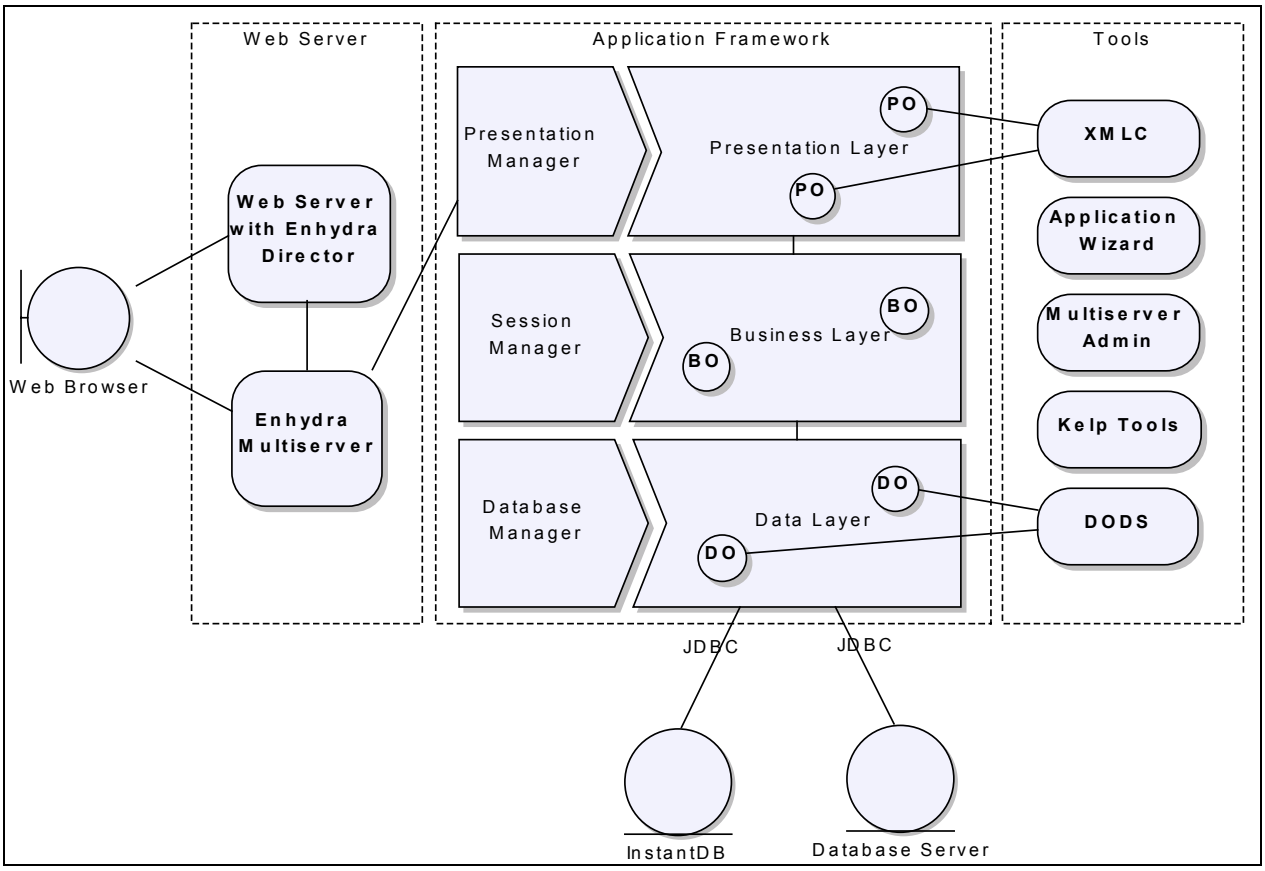


**Figure 2: Enhydra Architecture**

As illustrated in Figure 2, the Enhydra application server consists of the following three components:

- A Web server. The Web server can either be the Enhydra Multiserver, which is an HTTP 1.1 Web server with servlet engine supporting the servlet API version 2.2, or any other Web server with servlet engine. Enhydra applications work with any standards-compliant servlet runner. Figure 2 also shows Enhydra Director, which is a plug-in available for several Web servers to provide load balancing to application servers while maintaining session affinity.
- The Application Framework. Currently, Enhydra offers two distinct application models. The developer may either choose the Enhydra-specific “Enhydra super-servlet application framework”, or the “servlet application framework” that follows J2EE standards. The differences are described in further detail below.

- Tools. Enhydra offers a set of tools that help build a Web application. The most important tools are listed below.

Enhydra separates the application logic into three layers: The presentation layer, the business layer, and the data layer. Following this concept ensures that the designers can independently work on the presentation while software engineers work on the application logic. Providing three distinct layers also increases the maintainability of the code.

#### 2.3.3.3 The Enhydra Application Framework Models

- The Enhydra super-servlet application model contains a single servlet that manages all presentation objects (presentation objects generate the Web pages). This application controller servlet has to be derived from a class provided by the Enhydra architecture. A new presentation object is instantiated with each request.
- The servlet application model is the model described by Sun for so-called “Web applications”. With this model, several servlets are usually written to respond to the requests. The servlets are instantiated at start time of the application, and then executed many times in separate threads.

#### 2.3.3.4 Enhydra Tools

- The Application Wizard serves to generate an initial source tree for the development, including make files. While this could be done manually, it can definitely save time during the start-up phase of a project.
- The Extensible Markup Language Compiler XMLC is the tool that allows the clean separation of the user interface and the application logic. It is used to compile an HTML page into a Java class containing the whole page as a hierarchy following the W3C’s Document Object Model (DOM). The obtained Java class can then be programmatically modified by replacing nodes or attributes with dynamic content.

  This method has the following advantage: A designer can independently create a Web page with his/her design tool of choice. Then, the programmer compiles the HTML page and accesses the desired contents via attribute IDs.

- The Data Objects Design Studio is a tool to model the data tables of an application with a graphical user interface. It is capable of generating both SQL scripts and Java code.
- The Kelp tools are a set of utility programs that allow the configuration of Enhydra projects to work within IDE's such as Borland's JBuilder or Oracle's JDeveloper.

#### 2.3.3.5 Concluding Remarks on Enhydra

While this thesis focuses on Web application development with Tomcat, Enhydra seems to represent a very interesting alternative that should definitely be considered due to its prospective J2EE support, and the clean separation of the presentation and the business layer. Being a pure Java application, the deployment should be unproblematic in standards-compliant servlet engines.

### 2.3.4 Selected References


- Rich Bowen, Ken Coar; Sams Press; ***Apache Server Unleashed: Chapter 1***; 2000
  Contains an overview on the history and the features of Apache.

- Lutris Technologies; ***Getting Started with Lutris Enhydra: Overview***;
  http://www.lutris.com/documentation/lutris-enhydra/35/books/getting-started/overview.html;

- Robert L. Mitchell, Computerworld; ***Linux, Apache, Enhydra: Can Open Source Move Up?***
  http://www.computerworld.com/cwi/community/story/0,3201,NAV65-1797_STO59101,00.html;
  March 2001

- Reviews of the Enhydra Application Server on the Web:
  a. Michel de Bruijn; WebTechniques; ***An Application Server with No Strings Attached***;
     http://www.webtechniques.com/archives/2001/03/progrevu/; March 2001
  b. Anne-Sophie Karmel; TechMetrics; ***Enhydra 3.01: Product Profile***;
     http://www.techmetrix.com/trendmarkers/tmk1200/tmk1200-4.php3; December 2000
  c. Nicholas Petreley; InfoWorld; ***Arcane or Not, Enhydra Is a Dream Come True for the Java-Loving Servlet Developer***;
     http://www.infoworld.com/articles/op/xml/00/07/24/000724oppetreley.xml; July 2000

## 2.4 Database Systems

There are quite a few Open Source database systems available; MySQL and PostgreSQL are the most popular ones. For smaller applications, both database systems are a stable, no-cost alternative to commercial database systems; however, their architectures show distinct differences. Therefore, it is important to weigh the advantages and disadvantages of both systems, and select the database that is most applicable for a given purpose. The characteristics of both systems are outlined below.

MySQL is available at http://www.mysql.com, PostgreSQL at http://postgresql.readysetnet.com.

License information can be obtained at:

http://www.mysql.com/doc/L/i/Licensing_and_Support.html for MySQL,
http://postgresql.readysetnet.com/devel-corner/docs/postgres/ln1274.html for PostgreSQL.

| Feature | MySQL V3.23 | PostgreSQL V7.1 |
|---|---|---|
| License | MySQL Free Public License (very restrictive for Open Source standards; e.g. a license needs to be purchased for applications requiring MySQL to function) | License of the University of California (unproblematic licensing) |
| Kernel architecture | Multi-threaded allowing for utilization of multiple CPU's | Single-threaded |
| Transactions with rollbacks | Not supported | Supported |
| Views | Not supported | Supported |
| Sub-selects | Not supported | Supported |
| Foreign keys | Not supported | Supported |
| Extendable type system | Not supported | Supported |
| Stored procedures | Not supported | Supported |
| Size limitations:<br>Table size<br>Row size<br>Columns/table | <br>2GB to 8TB (operating system dependent)<br>65534 (without BLOB's)<br>3398 | <br>64TB (all O/S)<br>Unlimited<br>1600 |

**Table 3: Comparison of MySQL and PostgreSQL Features**

Generally, it can be stated that PostgreSQL currently offers a richer set of features in processing the stored data. MySQL does not support some critical features, such as safe transactions or foreign keys that are usually part of relational database management systems. However, MySQL offers more standard data types and functions according to ANSI SQL. Furthermore, the additional processing required for handling the transaction-safe tables in PostgreSQL gives MySQL a significant speed advantage for insert and update statements.

Other databases that have been made available under an Open Source License more recently include:

- InterBase and the spin-off Firebird, open sourced by Inprise/Borland (since July 2000)
- SAPDB, open sourced by SAP AG (since October 2000)

**Selected References**

- Tim Perdue; phpbuilder.com; ***MySQL and PostgreSQL Compared***;
  http://www.phpbuilder.com/columns/tim20000705.php3; July 2000
  A Comparison of MySQL and PostgreSQL features, performance, and stability.

- Tim Perdue; phpbuilder.com; ***Open Source Databases: As the Tables Turn***;
  http://www.phpbuilder.com/columns/tim20001112.php3; November 2000
  Follow-up article with extensive performance comparison of MySQL and PostgreSQL

## 2.5 Source Code Control with CVS

The creation process of software becomes more complex with growing size: An increasing number of developers have to collaborate closely to achieve effective development in larger systems. Furthermore, the introduction of latent bugs in the early stages of development can pose a real problem in large programs, since tracing the problem to its origin can become a tedious task.

Versioning control systems are the solution to this challenge. All versions of the source code are archived and maintained in a central repository (the “master” copy), usually at a separate

server. Developers are granted access to this repository only indirectly through formal check-in and checkout of code. The idea is that all developers work with their own local copy of source code, and only check their code in once it compiles and runs. Versioning control systems keep track of who is currently working on which files, and therefore prevent developers from accidentally deleting or overwriting one another's code. In addition to that, the repository keeps a complete history of all versions of the source code, so any version of a file may be retrieved later on, or the differences of any two versions can be analyzed.

The dominant versioning control system today is an Open Source Unix/Linux command line tool called Concurrent Versions System, or short CVS. It is widely used in both the Commercial and the Open Source software development world. CVS is based on RCS (Revision Control System).

Most Linux distributions ship with CVS. The versioning control system tools may also be downloaded from Web sites (e.g. http://www.cvshome.com) as source code, or as binary version for all common operating systems.

# 3 Introduction to Web Application Development

Chapter 2 gave an overview on current Open Source products and their features. This section deals more with the technical aspects of selected Web application development technologies, which should give the reader a good starting point for further study of the subjects.

The author assumes that the software packages described in this section have been successfully installed. Detailed installation instructions are available on the Internet for each of the products.

## 3.1 A Brief Introduction to Servlets and JSP

### 3.1.1 The Servlet Architecture

When a Web server receives a client request for static web pages (usually files with an extension *.htm* or *.html*), it looks for the requested page, and simply responds to the client with the contents. However, when dynamic content is requested, the behavior of the server needs to be different: In this case, a program has to be run on the server machine that interprets the parameters of the request and generates the content accordingly. To enable the server to distinguish between regular requests for static pages and requests for dynamic content, the server needs to be made aware of which requests should map to a program. This is usually done through a server plug-in. This plug-in is a small interface that tells the server which requests are dynamic (e.g. all pages that end with *.cgi* or *.jsp*), and what program to forward these requests to.

A popular way of generating dynamic content in the past was the Common Gateway Interface (CGI). The CGI environment instantiates a new child process on the server to handle each request, i.e. runs a program that generates the output dynamically. This method, however, is very resource intensive. Creating a the new runtime environment, initializing it and destroying it after use are overhead tasks that can weigh heavily when a Web server receives thousands of requests per day. A large number of simultaneous requests could even cause the server to crash due to memory restrictions.

Servlets offer a much lighter weight approach to handling many client requests. Although the servlet interface definition is based on CGI, it processes requests differently. A single program running on the server, a so-called Servlet Engine or Servlet Container, handles all requests. Instead of creating a new process for each request, the Servlet Engine uses Java's multithreading capabilities to generate the dynamic page within the Servlet Engine's main process. This approach results in a much better performance than the classical CGI approach. The interoperation between the Web server, the Servlet Container, and the client browser is illustrated in Figure 3.

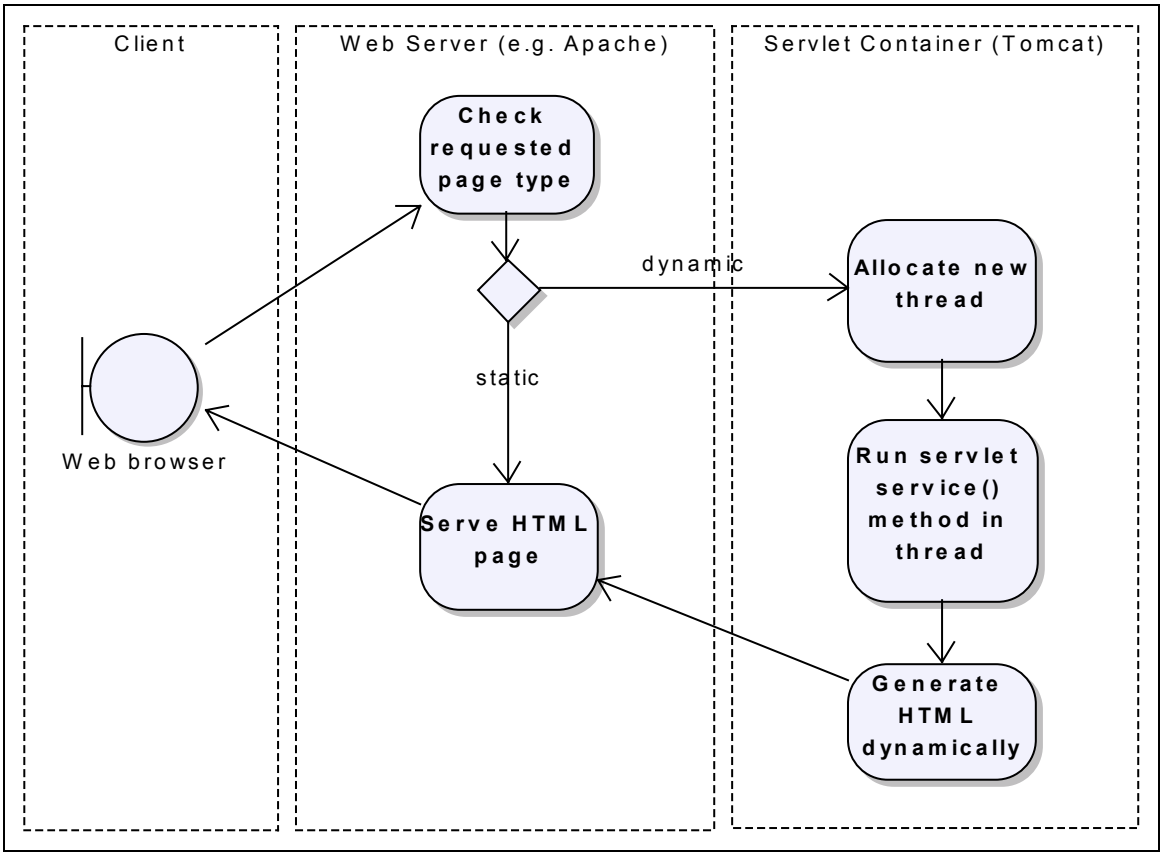


**Figure 3: Interoperation Client / Web Server / Servlet Container**

### 3.1.2 The Servlet Container

The Servlet Container, as illustrated by Figure 3, receives the HTTP request for a dynamic page from the Web server, and is then responsible for invoking the servlet and returning the generated response to the Web server. Exactly how the Servlet Container processes the requests can vary from implementation to implementation. However, an efficient and therefore common implementation of a Servlet Container is shown in Figure 4.

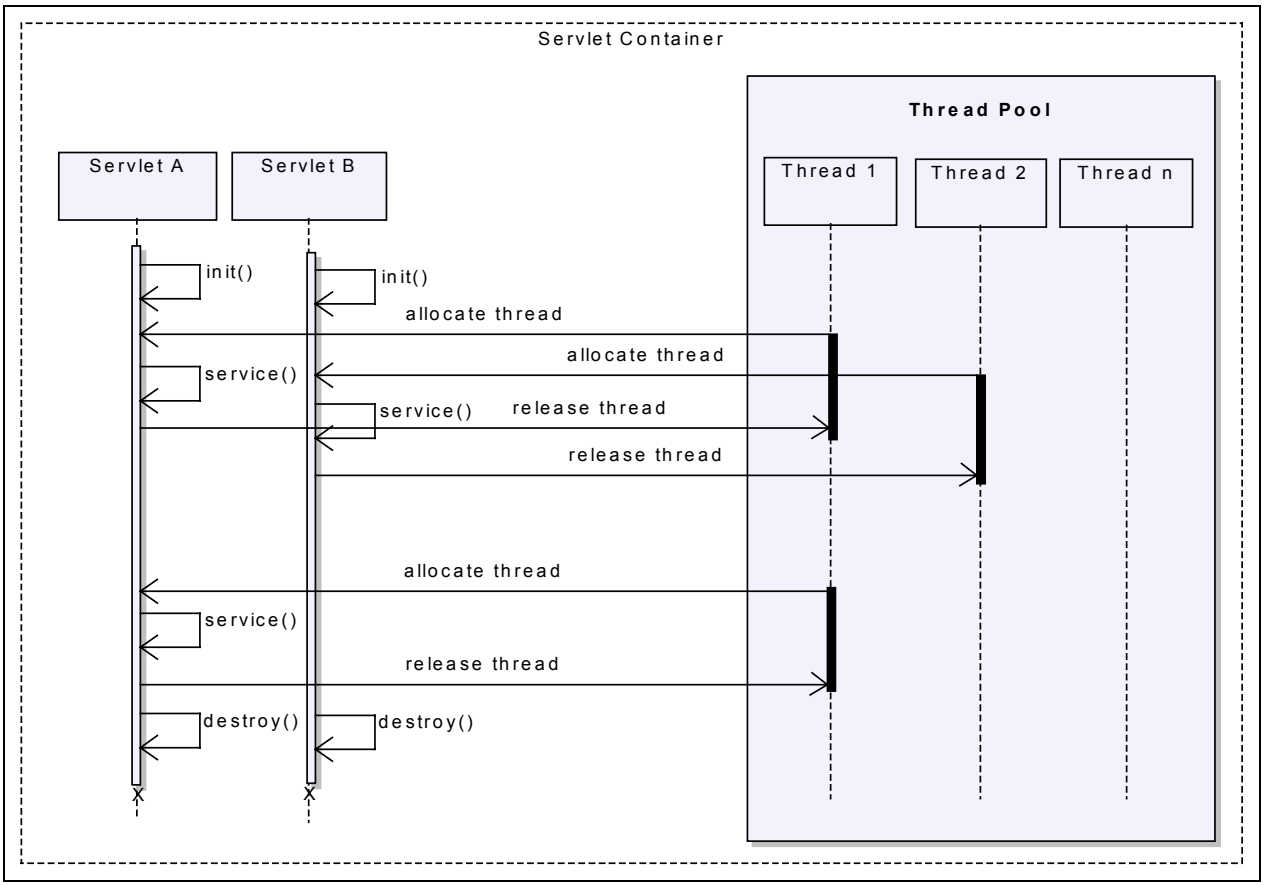


**Figure 4: Typical Servlet Container Implementation**

In this model, the Servlet Container creates a “Thread Pool” at start-up time. A Thread Pool is a collection of threads. When a new request is received, one of the available threads is allocated to the request. The thread then runs the *service()* method of the servlet that maps to the request. The thread is returned to the Thread Pool afterwards, so it can be re-used for another request. With this model, many requests can be handled simultaneously without the Servlet Container having to instantiate any new objects. The number of available threads in the Thread Pool is usually configurable, so performance can be optimized for the number of requests that an application usually receives over a certain time period. If all threads are in use, and another request is received, the synchronization capabilities of Java allow waiting for the next thread to become available.

A new instance of a servlet is only created when the servlet is requested for the first time. The Servlet container makes sure that the servlet’s initialization method *init()* has finished before the *service()* method is called. The servlet’s *service()* method can be run many times, and can even

be run simultaneously through several threads (there is no need for having a separate instance of a servlet object for each request; only a thread is directly associated with a request).

A servlet is only removed from memory if it has not been requested over a longer period of time. Prior to removal, the servlet's *destroy()* method is called to allow for clean-up activities.

### 3.1.3 An Example Servlet

Java servlets always look very similar. A servlet class usually contains just a few methods, the *init()*, a *service()*, and the *destroy()* method. If specific initialization or clean-up activities are not required, the *init()* and *destroy()* methods may even be omitted. An example for a simple, but typical servlet class is shown below. This servlet reads a few parameters from the client request, performs a task, and generates the HTML output.

```
package Rectangle;

// import the servlet packages
import javax.servlet.*;
import javax.servlet.http.*;

// import some other packages
import java.io.*;
import java.util.Arrays;

public class DrawRectangle extends HttpServlet {

  // doGet method is an HTTP specific service() method
  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

    // read two parameters from the request object
    int width, height;
    try {
      width = Integer.parseInt(req.getParameter("width"));
      height = Integer.parseInt(req.getParameter("height"));
    }
    catch (NumberFormatException e) { width = 10; height = 10; }

    // perform action: generate a page & a rectangle with * characters
    char line[] = new char[width];
    Arrays.fill(line,'*');
    StringBuffer str = new StringBuffer();
    str.append("<html><head><title>");
    str.append("Rectangle Test");
    str.append("</title></head><body><p>");
    for (int i=0; i<height; i++)
```

```
    {
      str.append(line);
      str.append("<br>");
    }
    str.append("The width is ");
    str.append(width);
    str.append(" and the height is ");
    str.append(height);
    str.append(".<br></body></html>");

    // write output to the response object
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println(str);
    out.close();
  }
}
```

The *service()* method (here called "*doGet()*") always has two arguments, which are an input and an output object. For an HTTP servlet, those objects are the HTTP request and the HTTP response. The HTTP request object passes all arguments to the servlet that were received by the Web server with the request. The response object contains a PrintWriter stream to hold the response that is generated by the servlet.

The simple servlet shown above reads two parameters *width* and *height* from the request object. Then, it draws a rectangle of these dimensions to a string buffer. Finally, it writes the string buffer containing the rectangle to the response object.

While this servlet does not seem particularly useful, and the same functionality could be achieved through some browser-side scripting, it does already show some of the servlet architecture's power: The *doGet()* method uses *fill(char[ ] a, char val)* of the Java *Arrays* package. All of Java's extensive library packages can be accessed through servlets, as well as any other classes or methods available on the server (custom classes, J2EE beans, etc.).

### 3.1.4 JavaServer Pages

JavaServer Pages (JSP) are a server-side extension to the servlet technology. A JavaServer Page is basically an HTML or XML page that contains scripting elements (scriptlets) and tags in addition to the regular page contents. A JSP engine always converts/compiles a page to a servlet prior to execution. JSP engines are included in most servlet engines, such as Tomcat.

JSP's offer the following advantages over pure servlet programming:

- JSP pages make it easier for a Web designer to generate pages with dynamic content. The Web designer can call so-called "JavaBeans" (for an explanation, see below) with XML-tag-style commands, which most Web designers are familiar with. No additional programming skills are required. Therefore, JSP pages foster the concept of separating application logic from content presentation.
- Web developers can use JSP pages for the generation of presentation-driven contents, such as complex forms or outputs that would be cumbersome to implement with servlet programming only.

JSP's use the following elements to extend the functionality of static HTML or XML pages with dynamic content:

- Directives (enclosed within `<%@` and `%>`) are short statements that are used to give the JSP engine information about the page that follows (such as page language, used packages, or whether the page should have session state). Directives also provide the means to include other HTML or JSP pages on the server side.
- Declaratives (enclosed within `<%!` and `%>`) are used for page-wide variable and method declarations.
- Scriptlets (enclosed within `<%` and `%>`) contain Java code embedded in the page. Unlike client side JavaScript or VBScript, Scriptlet code is interpreted on the server side.
- Expression Evaluations (enclosed within `<%=` and `%>`) offer a way to directly evaluate a variable or function to a String for inclusion in the output of the page.
- JavaBean invocation and manipulation tags (enclosed within `<jsp:` and `/>`). JavaBeans can be instantiated with the *useBean* command. In a nutshell, JavaBeans

are public Java classes with a constructor that has no arguments. They usually are reusable components containing application logic.

The following is an example JSP page demonstrating four of the five JSP language elements (the JSP elements are highlighted with a shaded background). This page uses a JavaBean to generate a formatted navigation bar.

```
<html>
  <head>
    <title>JSP Navigation Bar</title>
  </head>
  <body>
  <!-- JSP directive indicating the page language -->
  <%@ page language="java" %>
  <!-- JSP directive indicating a server-side page include. -->
  <%@ include file="banner.html" %>
  <!-- Use a JavaBean to generate a navigation bar -->
  <jsp:useBean id="navibar" class="MyPackage.NavigationBar" />
  <!-- Now, call the navibar bean's add() method to manipulate the bean.
     This is done in Java language syntax. The following two lines are
     a "scriptlet". -->
  <% navibar.add("Home","home_int.jsp");
     navibar.add("Help","help_int.jsp"); %>
  <!-- Evaluate an expression with <%= . . . %>, in this case, call the
    navibar bean's toHTML() method to generate the navigation bar in HTML
    on the page -->
  <%= navibar.toHTML() %>
  <h1>Welcome to this demonstration page!</h1>
  <p>This page shows how to use JSP to generate a navigation bar.</p>
  </body>
</html>
```

### 3.1.5 Selected References


- Danny Ayers et al; Wrox Press Ltd; ***Professional Java Server Programming***; 1999
- Sun Microsystems, Inc.; ***Java Servlet Technology, White Paper***; http://java.sun.com/products/servlet/whitepaper.html
- Sun Microsystems, Inc.; ***JavaServer Pages, White Paper***; http://java.sun.com/products/jsp/whitepaper.html

## 3.2 Introduction to Web Application Development with Tomcat

The Apache Software Foundation has published an excellent guide on application development under Tomcat at http://jakarta.apache.org/tomcat/jakarta-tomcat/src/doc/appdev/. The author strongly recommends basing a new development environment on the instructions provided by this document. This Section provides a brief summary of the key ideas of Apache's application development guide, some additional comments for clarification, and a detailed example for creating a simple application, which can serve as a starting point for more complex tasks.

**Important note**: Depending on your user account settings on the system you are using, you might need to have Linux super-user (administrator) rights to write to directories that are not inside your local user home directory. You may also require logging in as a super-user to configure the Tomcat servlet engine, or to start and stop it during development. If you are not familiar with the access rights and user management on your Linux system, please contact your system administrator. The system administrator may also wish to grant you the required permissions explicitly by providing the appropriate *sudo* commands.

### 3.2.1 Creating the Source Code Repository

The source code should be kept separate from the compiled version of the application. Source code control is easier if the directories contain only source files, and it is much more convenient to make an installable distribution of the application. The Apache Software Foundation recommends a directory hierarchy for source files, which is shown in Figure 5.

Using the recommended directory structure also has the advantage that a provided build script may be used without major modifications, which can create a compiled Web application from the source tree with just one shell command.

Before creating the source repository with the Concurrent Versions System, CVS needs to be configured according to the access levels required. CVS repositories can be configured for single users, user groups on the same server, or for multiple users on many client machines using remote access over a network or the Internet. Described here is the configuration for a user group accessing the repository from a single server. Please see Section 3.2.5 for references on how to set up CVS for the other cases.

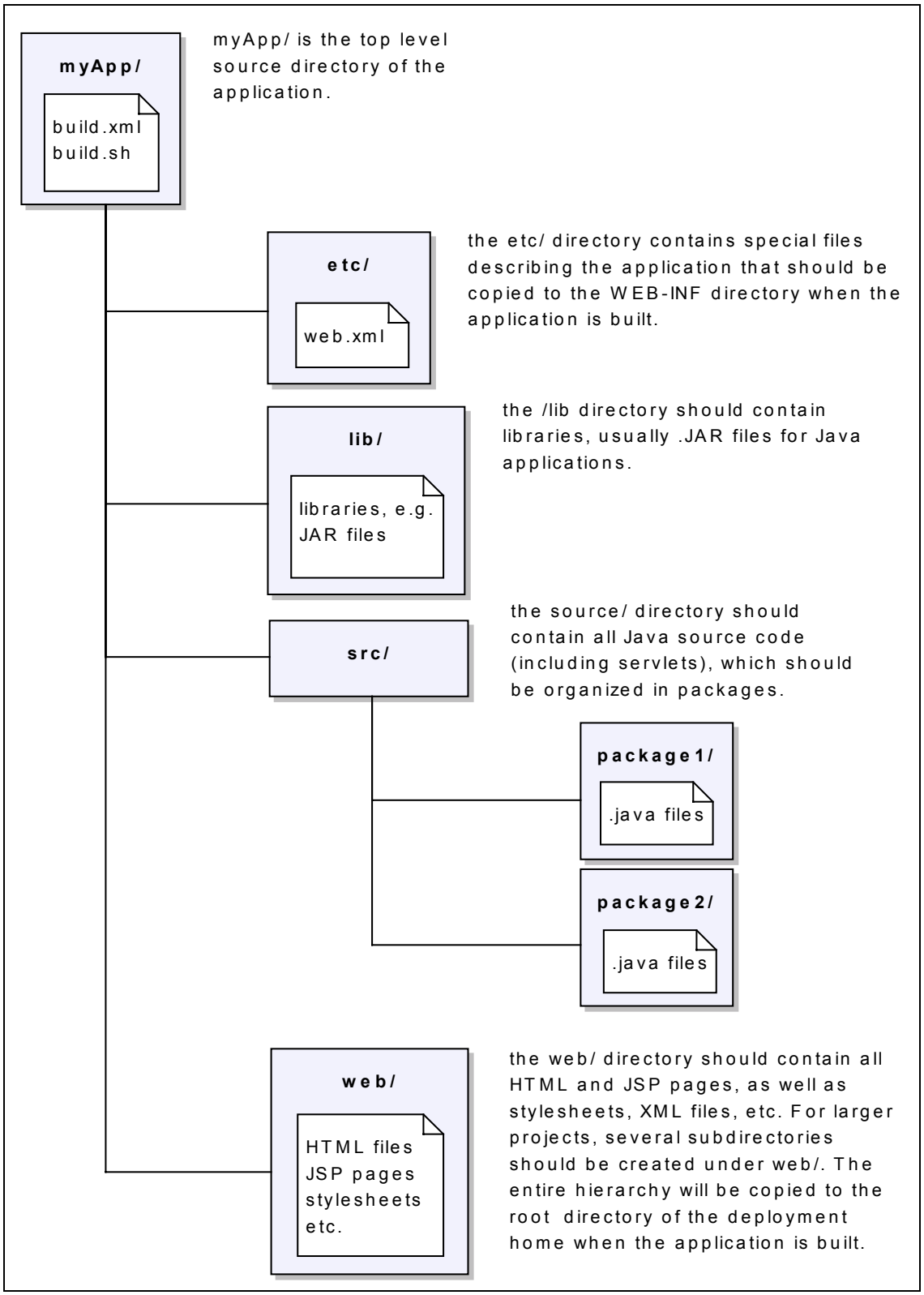


**Figure 5: Source Code Directory Structure**

#### 3.2.1.1 Configuring CVS for a User Group on a Single Server under Linux

To set up CVS from the Linux shell for multiple developers within a user group on the same server, you may follow the steps outlined below. The $ symbol stands for the Linux shell command prompt.

- Step 1: Create a Linux user group that all developers will belong to (assuming that the prospective developers already have login accounts), and then add the users to this new group. Note: You need to have super-user privileges to create new groups.

```
$ /usr/sbin/groupadd devteam              (create new group "devteam")
$ /usr/sbin/usermod -G devteam andreas    (add user "andreas" to devteam)
$ /usr/sbin/usermod -G devteam grace      (add user "grace" to devteam)
```

- Step 2: Set the CVS environment variables. The *export* statement is used to set system variables under Linux. The following statements could be added to the global */etc/profile* file or the users' *.profile* file to set the variables automatically at login time.

  First, set the default editor for CVS. You may use vi, ae, emacs, or any other editor. CVS invokes the editor when an entry to a log file is required.

```
$ export EDITOR=/bin/vi
```

  Next, set the CVS root directory. Make sure you have write permission in the directory where you wish to place the CVS root.

```
$ export CVSROOT=/devteam/cvsroot
```

- Step 3: Create the CVS root directory with the following command. The $ symbol indicates that the following text is a system variable (the one you have defined in Step 2).

```
$ mkdir $CVSROOT
```

- Step 4: Set the appropriate group ownership and access permissions for the repository.

  First, set the group for the new CVS repository:

```
$ chgrp devteam $CVSROOT
```

  Then, change the access permissions of the directory to allow all members of the development group to read from and write to the new CVS repository.

```
$ chmod g+rwx $CVSROOT
```

- Step 5: Initialize CVS with the command

```
$ cvs init
```

#### 3.2.1.2 Creating the Source Tree in CVS and Configuring the Build Scripts

This Section describes how to set up the source tree shown in Figure 5 within the CVS repository created under Section 3.2.1.1.

- Step 1: Before you create the directory tree in CVS, you need to create a local copy of the directory structure in your home directory.

```
$ cd ~                (The ~ symbol is a variable containing the current
$ mkdir myApp          user's home directory)
$ cd myApp
$ mkdir etc
$ mkdir lib
$ mkdir src
$ mkdir web
```

- Step 2: Create the initial project in CVS using the *cvs import* command. Replace *{username}* with your Linux user name.

```
$ cvs import -m "Initial project creation" myApp {username} start
```

  You can verify that the project was created correctly in CVS by renaming the original directory to keep it as backup, and performing a checkout:

```
$ cd ..
$ mv myApp myApp.bu
$ cvs checkout myApp
```

- Step 3: Create and customize the build scripts for your application. As mentioned earlier, the Apache Software Foundation already provides build files as a basis. These files require the Apache "Ant" make utility to run, which is usually included in the Tomcat distribution and does not need to be installed separately. However, if Ant is not yet installed on your system, you may download it from http://jakarta.apache.org/ant/ (Documentation is available at this Web address, too).

  Download the files *build.xml* and *build.sh*, and place them in your local *myApp/* directory. Links to the files can be found within the document http://jakarta.apache.org/tomcat/jakarta-tomcat/src/doc/appdev/.

You don't need to make any modifications to the *build.sh* file.

The *build.xml* file, however, must be customized slightly so it can be used by your application. Fortunately, you need to modify only a few lines to change the application name and to set the destination path indicating where Ant should deploy the application. You can make the changes by opening the *build.xml* file with a text editor, such as *emacs*.

The following lines give an example for the modification that is consistent with the folder names introduced above.

```
<property name="app.name" value="myApp"/>
<property name="deploy.home" value="/devteam/deployHome"/>
```

**Note:** The *build.xml* file contains detailed information on how to customize the file.

Then, add the initial build files to the source code repository from your local *myApp/* directory with the following commands:

```
$ cd ~/myApp
$ cvs add build.xml
$ chmod a+x build.sh    (this command makes the build script executable)
$ cvs add build.sh
$ cvs commit -m "Initial build script files added."
```

With the completion of Step 3, the preparation of the source code repository is complete. All developers can now check out the source code with *cvs co {application name}*, add new files to the repository with *cvs add* and *cvs commit*, or build the application using the *build.sh* script. The last part of the set-up process is the configuration of Tomcat and the application itself, so Tomcat can recognize it, and the application can be accessed through web browsers once it is built.

### 3.2.2 Configuring Tomcat to Recognize the New Application

To make the Tomcat servlet engine aware of the new application, you have to add a new *<context>* entry to Tomcat's *server.xml* configuration file with information on the location of your application. Furthermore, the application itself has to be deployed with a configuration file called *web.xml* describing the servlets it contains, as well as other critical application data. The necessary steps for a successful set-up of the two files are outlined below.

- Part 1: Configuring the Tomcat servlet engine's configuration file called *server.xml*.
  Open the *server.xml* file in a text editor, e.g. emacs.
  ```
  $ emacs $TOMCAT_HOME/conf/server.xml
  ```

  The *server.xml* file already contains some context elements. Navigate to the end of the file, and add the new context element, as shown below:

```
<Context path="/myApp"
   docBase="/devteam/deployHome"
   debug="0"
   reloadable="true"
   trusted="false" >
</Context>
```

  The *path* attribute indicates the virtual directory under which the browser may access the physical document root directory *docBase* on the server (this should be the directory containing the web application – during development, it should be the same directory as the deployment home directory specified in the *build.xml* file).

  Please refer to the Apache Software Foundation's Tomcat application development guide for a detailed explanation of the other attributes.

- Part 2: Creating an initial *web.xml* file for the new application.
  The *web.xml* file is the Web Application Deployment Descriptor. This file defines all information that the server needs to know to run the Web application. The definition of the Descriptor is part of the Servlet API Specification, V2.2. The Apache Software Foundation provides a basic *web.xml* file within the Tomcat application development guide. Please download this file, and place it in your local source code directory under *etc/*. Then, add the file to the repository with the commands given below. As long as you have not created any new servlets, no modifications should be necessary to the provided *web.xml* file. However, the web.xml file contains a few small bugs that must be taken care of (as of the writing of this on April 30, 2001 – I am assuming they will still be there):

    - Line 77: The description text contains an XML tag <*servlet-mapping*> that is recognized by Tomcat, but should not be right here. Just remove the <> symbols.
    - Line 88 and 92: The end tags are wrong. Instead of <*/paramName*>, it should be <*/param-name*>

  Note: The example *web.xml* file contains a lot of information that is not required initially. Nevertheless, it is very useful to have an example that contains many of the available tags for reference.

  After correcting the mistakes mentioned above, you should add the file to the repository.

```
$ cd ~/myApp/etc
$ cvs add web.xml
$ cvs commit -m "Initial web.xml file added."
```

This concludes the set-up and configuration of the development environment. In the next section, the author will demonstrate the development and deployment of a simple application containing one HTML page and one servlet, and look at the structure of the deployed application in further detail.

### 3.2.3 Developing and Running an Example Application

The example application will consist of the Servlet shown in Section 3.1.3, and an HTML form that is used to provide the Servlet with the necessary parameters to generate some dynamic output.

#### 3.2.3.1 Creating the HTML Page and Registering It with the Repository

The source code for the HTML page is shown below. Store the code under the name *draw_rectangle.html* in the *web/* directory of the source tree. The form draw_rectangle.html is a plain HTML page; it does not contain any dynamically generated data. It serves only to provide the servlet with the necessary parameters to run.

```
<html>
  <head>
    <title> Draw a Rectangle </title>
  </head>
  <body>
    <h2>Servlet Demonstration - Draw a Rectangle</h2>
    <!-- The following line specifies the URL that the form is submitted to
         when the "Submit" button is pressed. The URL is mapped to the
         correct Java class by the Servlet Engine -->
    <form action="draw_rectangle.servlet">
      <p>
        Please enter the desired width and height:<br>
        Width: <input type=text name="width" size=5><br>
        Height: <input type=text name="height" size=5><br><br>
        <input type=submit value="Draw Rectangle">
      </p>
    </form>
  </body>
</html>
```

Schedule the *draw_rectangle.html* file for addition to the repository with the following command:

```
$ cvs add draw_rectangle.html
```

Please note that the *cvs add* command does not yet store the file in the repository. This gives you the opportunity to test the file before committing it to the repository. Committing the file will be handled later on in Section 3.2.3.7.

#### 3.2.3.2 Adding the Servlet to the Repository

Create a new directory under the *src/* directory with the name *Rectangle*. The directory name must be the same as the package name specified in the first line of code of the servlet. Then, save the Servlet code presented in Section 3.1.3 under its class name (*DrawRectangle.java*) in the *Rectangle* directory. Schedule the new directory (this is important: always add directories to the repository as well!) and the new file for addition to the CVS repository.

```
$ cd src
$ mkdir Rectangle
$ cvs add Rectangle
$ cd Rectangle
$ cvs add DrawRectangle.java
```

#### 3.2.3.3 Registering the Servlet in *web.xml*

Before the new servlet can be executed, it needs to be registered in the Web Application Deployment Descriptor *web.xml*.
To do this, add the following lines to the *web.xml* file:

```
<servlet>
  <servlet-name>DrawRectangle</servlet-name>
  <servlet-class>Rectangle.DrawRectangle</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>DrawRectangle</servlet-name>
  <url-pattern>draw_rectangle.servlet</url-pattern>
</servlet-mapping>
```

The <*servlet*> tag defines a new servlet. You must provide a servlet name, which can be chosen arbitrarily (it does not necessarily have to be the servlet class name). The <*servlet-class*> attribute tells the servlet engine where to find the java class file that contains the servlet byte code. Last but not least, you need to define the servlet mapping. (The URL that is received through the browser request is mapped to a Java class by the servlet engine).

A servlet mapping should be provided for each servlet. You may also use wildcards for the URL-patterns. E.g. <*url-pattern*>*.*servlet*</*url-pattern*> would map all URL's ending with *.servlet* to the specified servlet name.

#### 3.2.3.4 Building the Application

To build the application, please run the *build.sh* script from your local source code root directory *myApp*.

```
$ cd ~/myApp
$ ./build.sh
```

The build script will automatically determine which files have to be re-compiled or re-copied to the deployment directory that you have specified in the *build.xml* file. When you use *build.sh* the first time, the make utility Ant will create the whole directory structure from scratch.

The *build.xml* file and the source code directories are organized in a way that the Servlet API Specification V2.2–compliant Web application hierarchy is generated automatically. This standard directory layout is shown in Figure 6.

When building the application, you may also specify an optional parameter to control the build behavior (`$ ./build.sh [parameter]`). The available options are shown in the table below.

| Option | Function |
|---|---|
| `prepare` | Generates the deployment directory structure. |
| `clean` | Removes the deployment home directory structure. |
| `compile` | The default; Compiles/copies the project to the deployment home. |
| `javadoc` | Generates the Java documentation. |
| `all` | Executes clean, prepare, compile, and javadoc consecutively to re-build all. |
| `dist` | Generates the WAR file for the application; a Web archive is a compressed file that contains the whole Web application in a single file, conforming to the Servlet API Specification V2.2. The Web archive is used to easily distribute and deploy an application. The dist command also generates a JAR file of the source code. |

**Table 4: Application Build Options**

Eventually, you may wish to modify the provided *build.xml* file, for example to generate the Java documentation in another directory, or to change the *javac* compiler settings (turn optimization on, for example). The Apache Ant documentation explains the available XML tags and attributes that make up the build script in the Ant reference documentation, which can be found here: http://jakarta.apache.org/ant/manual/.

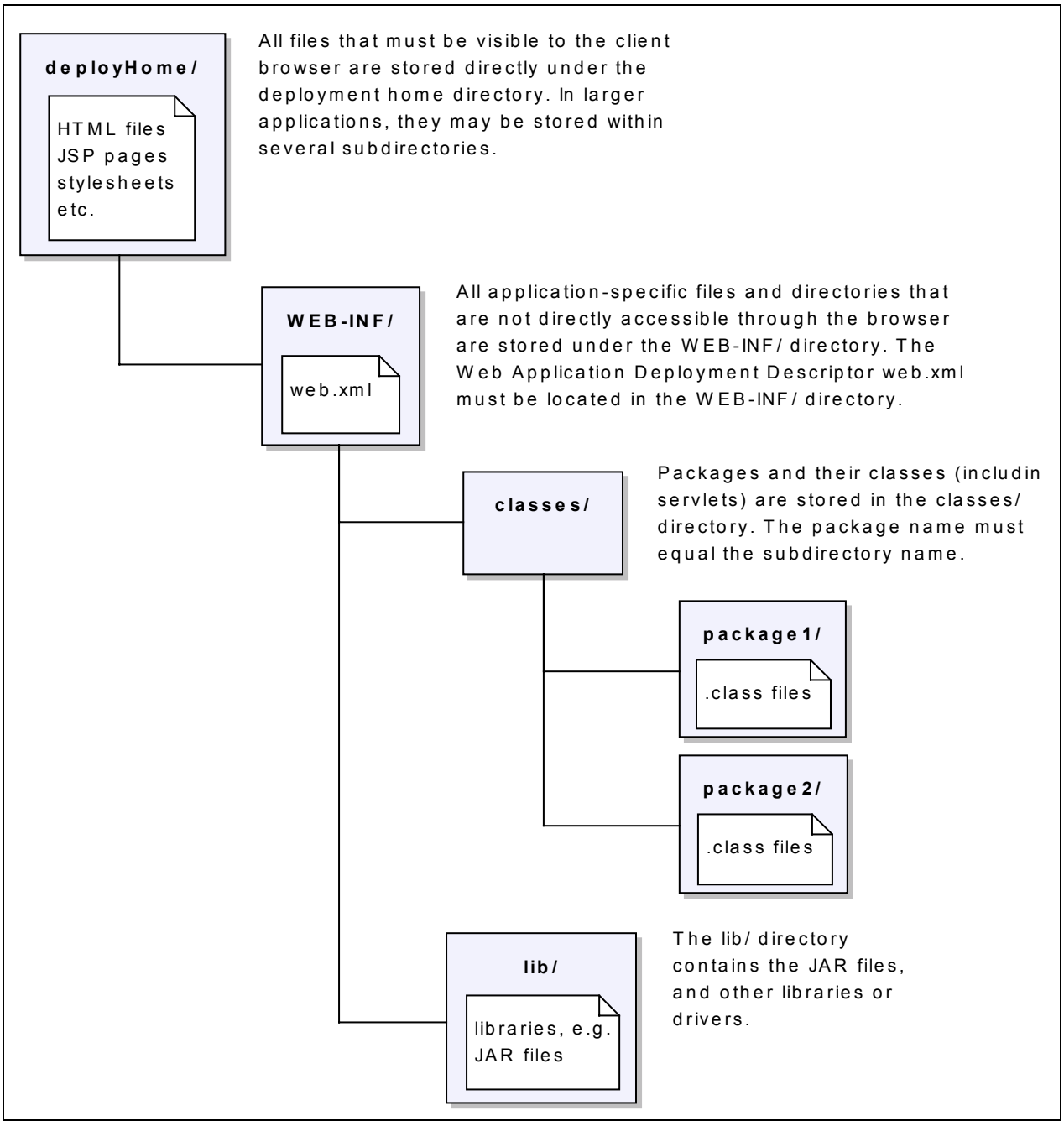


**Figure 6: Standard Directory Layout, Servlet API Specification V2.2**

#### 3.2.3.5 Starting and Stopping Tomcat

Once you have successfully built the application, you can run it in the servlet engine. The command to start Tomcat is simply:

```
$ $TOMCAT_HOME/bin/startup.sh
```

You should receive messages indicating the start-up process, similar to this:

```
Using classpath:/usr/local/jakarta-tomcat-3.2.1/lib/ant.jar:
/usr/local/jakarta-tomcat-3.2.1/lib/jasper.jar:
/usr/local/jakarta-tomcat-3.2.1/lib/jaxp.jar:
/usr/local/jakarta-tomcat-3.2.1/lib/parser.jar:
/usr/local/jakarta-tomcat-3.2.1/lib/servlet.jar:
/usr/local/jakarta-tomcat-3.2.1/lib/test:
/usr/local/jakarta-tomcat3.2.1/lib/webserver.jar:
/usr/java/jdk1.3/lib/tools.jar:
/usr/local/jakarta-tomcat-3.2.1/webapps/ROOT/WEB-INF/classes:
2001-05-01 12:10:39 - ContextManager: Adding context Ctx( /examples )
2001-05-01 12:10:40 - ContextManager: Adding context Ctx( /admin )
2001-05-01 12:10:40 - ContextManager: Adding context Ctx( /myApp )
cannot load servlet name: controller
2001-05-01 12:10:42 - PoolTcpConnector: Starting HttpConnectionHandler on
8080
```

Tomcat is pre-configured to listen and respond to HTTP requests at port 8080. You can verify this at the start-up, as shown above (last line). There should also be a line indicating that your application context was loaded (other bold line).
Don't worry about the message *"cannot load servlet name: controller"*, which you will probably get. This is a servlet defined in the *web.xml* file, but there is actually no class provided for it in our example application; therefore, it cannot be loaded, of course. You can clean up *web.xml* and remove this servlet, as well as any other definitions not required later on.

Any time that you modify the *web.xml* file (by adding a new servlet, for example), Tomcat has to be restarted so it reloads the Web Application Deployment Descriptor *web.xml* (However, you don't have to restart Tomcat when adding JSP pages or HTML pages, since they do not require a modification of *web.xml*). Before running the start-up script again, you have to shut Tomcat down properly, or Tomcat will not restart. To do this, use the command:

```
$ $TOMCAT_HOME/bin/shutdown.sh
```

When continuously changing and deploying the application, you may wish to write a batch script that stops and restarts Tomcat with a single command.

### 3.2.3.6 Testing the Application

To test the application, open a Web browser, such as Microsoft Internet Explorer or Netscape Navigator. If you have used the same settings as described in the previous sections, you can access the application under the URI http://localhost:8080/myApp/ from your Tomcat server. From any other machine, you have to replace the *localhost* keyword with the Web server name (e.g. *myServer.mit.edu*) or the server's IP address.

By pre-definition, browsing of the application directories is allowed (you can disable this later, if you wish). However, the WEB-INF directory (see Figure 6), which contains the application configuration file and the servlets, is not accessible to the client browsers.


**Figure 7: Browsing the Application's Folders**

To go to the example page, click on the file draw_rectangle.html (the only file directly accessible from a browser). The following figures show the HTML page and the servlet output.

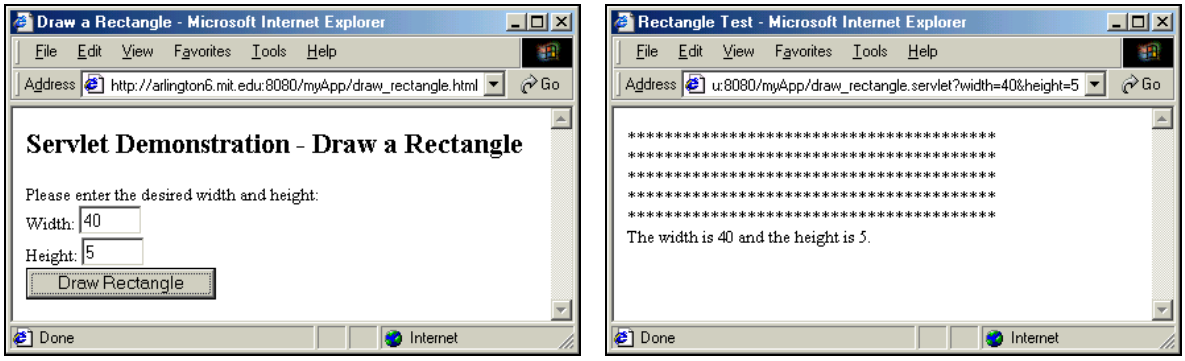


**Figure 8: Example Application Screenshots**

#### 3.2.3.7 Committing the Additions to CVS

When you have tested the pages, and everything runs to your satisfaction, you should finally commit the new files and the changes you have made to the CVS repository.

```
$ cd ~/myApp
$ cvs commit –m "Added draw_rectangle files & modified web.xml file."
```

Good practice is to build the application and check for errors before committing any changes. This way, the repository will contain a running version at all times. This is advantageous especially when multiple developers are involved in the project.

You can always issue the *add* command well ahead of the actual commit, as demonstrated with the two pages of the example application. By adding the files early, you can reduce the danger of forgetting to check in a file. The *cvs add* command will only schedule a file for committing, but not actually store it in the source code repository yet. The *cvs commit* command will update the repository.

### 3.2.4 Concluding the Introduction to Application Development with Tomcat

Section 3.2 has given a detailed introduction to setting up a development environment for Web applications using Tomcat and CVS on the Linux operating system.

Most tasks have to be performed manually. While the process of registering servlets, managing CVS, and building the application with a make file seem tedious compared to the development with IDE's, the demonstrated approach gives the development team good control over the product, as well as flexibility. However, the author encourages the reader to consider other development environments, such as Enhydra, which provide a richer set of tools to create a Web application faster (e.g. wizards to set up a source tree, integration with IDE's).

#### 3.2.4.1 Debugging with Tomcat

A difficulty that the developer faces is the lack of a real debugging tool for servlets. Since a servlet can be tested only from a browser, and stepping through a servlet line-by-line is therefore not possible, debugging can become a time consuming task. Therefore, you should apply extra care when developing a servlet, and make good use of status outputs with the *System.err.println()* command, for example. There are IDE's available that can be used with Tomcat to improve debugging issues, such as Borland's JBuilder. Please refer to Borland's Web site at http://community.borland.com/article/0,1410,22057,00.html for additional information (JBuilder is not Open Source software, but the Foundation Edition V3.5 is available free of charge).

#### 3.2.4.2 Multiple Developers with CVS

The small size of the Web application example presented above did not really allow for showing the advantages of the central CVS repository that was created for the application source code. The following is a brief explanation of how other developers can access the code.

Any member of the development team that is part of the group owning the repository can checkout the code to his/her home directory and independently work with it. A user *x* would simply use the following commands to obtain a fresh local copy:

```
$ cd ~
$ cvs co myApp
```

Now, the user *x* can edit the files, add new files, commit changes, or build the application just like the person that created the initial version.

To bring a checked-out version up-to-date with the current state of the repository, the *cvs update* command is used from the project source directory:

```
$ cd ~/myApp
$ cvs update -d
```

It is usually helpful to coordinate among the developers and decide who should work on which files to avoid two developers checking out the same file and working on it simultaneously (files that are checked out are not locked by CVS). Although CVS has merging capabilities to bring different versions together, this does not work very well. It is also recommended to commit changes and run *cvs update* regularly to avoid conflicts that can occur when developers use older versions of files.

CVS offers many other features. Please refer to the resources given in Section 3.2.5 for additional information.

#### 3.2.4.3 Scalability of the Presented Development Approach

The presented approach installed Tomcat and the source code repository on one central server. Building and deployment of the application was also done on this server. This works well for smaller development teams of around three team members. The developers can access the central server easily via Telnet, and have their own local copies of the source code in their home directories.

However, concentrating the development on a single server has disadvantages. Assuming that all developers use the same deployment directory for their builds and compile their source code on the same system, a larger number would cause too much interference among them. Stopping and restarting Tomcat, which has to be done occasionally to re-load the application, as well as limited system resources slow down development as the number of developers increases.

The easiest and most efficient way of addressing these issues is to provide each team member with his/her own machine running its own local instance of Tomcat. Other options are available as well, which will be introduced in Section 5.2.

### 3.2.5 Selected References

- Mark Allen; ***The CTDP Linux User's Guide V0.6.2***; November 2000;
  http://ctdp.tripod.com/os/linux/usersguide/index.html
  A good Linux user's guide.

- Nathan Meyers; Waite Group Press; ***Java Programming on Linux: Chapter 9: Setting Up a Linux Development Environment***; 2000;
  Not specifically for Web applications, but contains an introduction to CVS

- Apache Software Foundation; ***Developing Applications with Tomcat***; 2000;
  http://jakarta.apache.org/tomcat/jakarta-tomcat/src/doc/appdev/
  This guide is essential for getting started with Web application development under Tomcat.

- Alavoor Vasudevan; ***CVS-RCS How-To V19.0***; January 2001;
  http://linuxdoc.org/HOWTO/CVS-RCS-HOWTO.html
  A practical guide to very quickly set up the RCS/CVS source code control system.

- John Goerzen; IDG Books Worldwide, Inc.; ***Linux Programming Bible: Chapter 26: Archiving and Collaboration with CVS***; 2000;
  This chapter contains information on how to set up CVS for single users and for multiple users via remote access. Also contains a brief description of the most used CVS commands.

- Karl Vogel; Coriolis Inc.; ***Open Source Development with CVS***;
  Chapters 2, 4, 6, 8, 9, and 10 are available free under the GPL at
  http://cvsbook.red-bean.com/cvsbook.html; 2000

# 4 Case Study: An Automated Street Opening Permit System

## 4.1 Introduction to the System

### 4.1.1 Overview and Purpose

The Street Opening Permit System was developed for the Engineering Division of the Department of Public Works of the Town of Arlington, Massachusetts, to improve the process of applying for, issuing of, and administering of permits for street openings. Moving from a manual process to a computer-supported approach promised good time savings both on the applicant's and the town's side. The system was implemented as an n-tier Web application consisting of a browser-based user interface, business logic programmed in Java, and a database backbone.

The core part of the system is an application form that is processed automatically right at submittal time, and makes an immediate decision whether a permit can be granted or not. To make this possible, the engineering department is given extensive administrative functionality to configure the behavior of the system. Furthermore, a database is used to archive all issued permits, as well as all other system-critical information.

The system distinguishes two different user profiles: External users, which are the contractors and utility companies that apply for permits, and internal users, which are the town's employees that manage the system.

Typical tasks for external users with the system are:

- Applying for permits
- Searching for, displaying and printing permits that have been issued to them

Typical tasks for internal users with the system are:

- Viewing, editing, deleting, or searching for issued permits
- Configuring the behavior of the automated application approval system
- Administering external users (contractors and utility companies)
- Generating reports or billing information

The application was developed under Tomcat on a Linux server. MySQL was used as the database. The system will also be deployed under the same software components due to the low expected load of the application, and the financial limitations that apply to public institutions.

### 4.1.2 Overall Architecture

The Permit System consists of three layers:

- The user interface, which was implemented using Sun's JavaServer Pages (JSP) technology. For presentation, both JSP and servlets are used.
- The business logic, which consists of a package of Java classes.
- The database layer, which consists of a set of tables in a MySQL database.

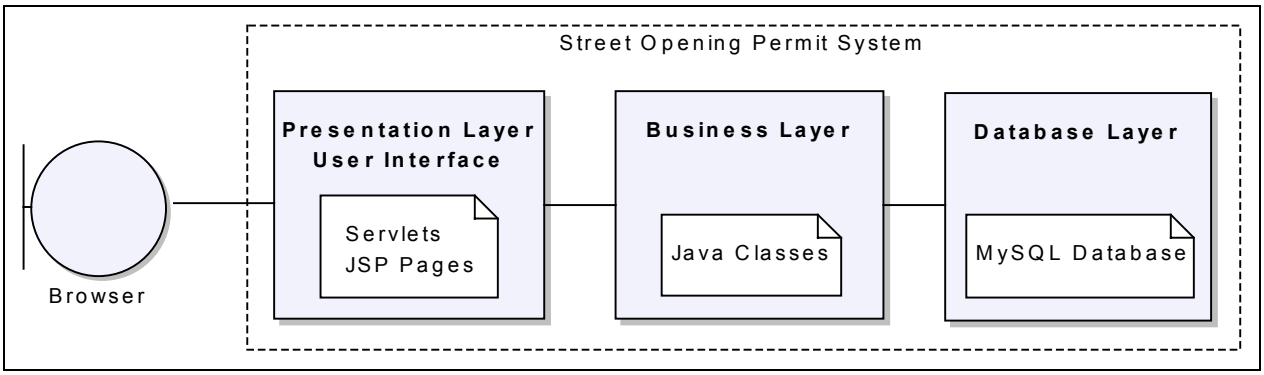


**Figure 9: Permit System Architecture**

The first step in developing the Permit System architecture was the design of the database by capturing the requirements as attributes and tables in a relational system. Then, an initial prototype of the user interface was created that consisted of HTML mock-up pages. The Java classes comprising the business logic were then developed closely based on the database tables and the HTML pages.

During the architectural process, a strong emphasis was placed on the consistency of the three layers. Fields of forms that were developed needed to have a corresponding attribute in a database table, and vice versa. Furthermore, a file name, attribute, and variable naming convention spanning all three layers was introduced to avoid confusion and to make the names easier to memorize.

## 4.2 User Interface

### 4.2.1 UI Design Decisions

When designing the user interface, a couple of items have to be kept in mind to arrive at a good solution. Basically, the system should be as easy and intuitive as possible to use. Therefore, the pages should be consistent in look and feel, which helps the user in getting accustomed to the system faster. Furthermore, the pages and functions should be logically grouped to facilitate navigation between the pages, and functions should be placed where the user would most likely expect them without having to consult a manual first.

To achieve these goals, the following design decisions were made:

- All pages have an identical layout. On top of the page, a banner is shown. Below that, a navigation bar is displayed from which the Home and the Help page may always be reached. Underneath the navigation bar, the main page area is displayed. Finally, at the bottom of the page, a disclaimer with a copyright notice and contact information is shown. The page banner and the disclaimer are included from an external file so updating this information can be done easily by modify only one file.
- Fonts and colors are consistent from page to page. This is done with a central Cascading Style Sheet that is included at the top of each page. CSS style sheets are a very powerful but simple way of reducing the formatting overhead on pages, and to give Web pages a uniform look across different browsers (Netscape, Microsoft). The style sheet that was used for the Permit System is shown below.

```
<!-- define the hyperlink style -->
a:link    { color: #000080; font-family: verdana, sans-serif;
            font-weight: bold }
a:visited { color: #000080; font-family: verdana, sans-serif;
            font-weight: bold }
a:active  { color: #000080; font-family: verdana, sans-serif;
            font-weight: bold }
a:hover   { color: #D00000; font-family: verdana, sans-serif;
            font-weight: bold }

<!-- define the body style -->
body { font-family :verdana,sans-serif; font-size: 10pt;
       background-color: #FFFFFF; color: #000000 }

<!-- define table and paragraph defaults -->
p,td,th { font-family : verdana, sans-serif; font-size: 10pt }
```

```
<!-- define the headings -->
h4 {font-family:verdana,sans-serif;
    font-size:10pt;color:#000080; text-align:center }
h3 {font-family:verdana,sans-serif;
    font-size:12pt;color:#000080; text-align:center }
h2 {font-family:verdana,sans-serif;
    font-size:16pt;color:#000080; text-align:center }
h1 {font-family:verdana,sans-serif;
    font-size:20pt;color:#000080; text-align:center }
```

- The internal user administration pages and the external user pages where clearly separated to avoid any implementation difficulties with respect to security of the internal Web pages. Mixing the external and internal content would have increased the complexity of providing an "employees-only" context area.
- All functions of the system were logically grouped and a page hierarchy was developed to have a clearly structured and consistent navigation. The external Web page hierarchy is shown in Figure 10. The internal Web page hierarchy is shown in Figure 11.

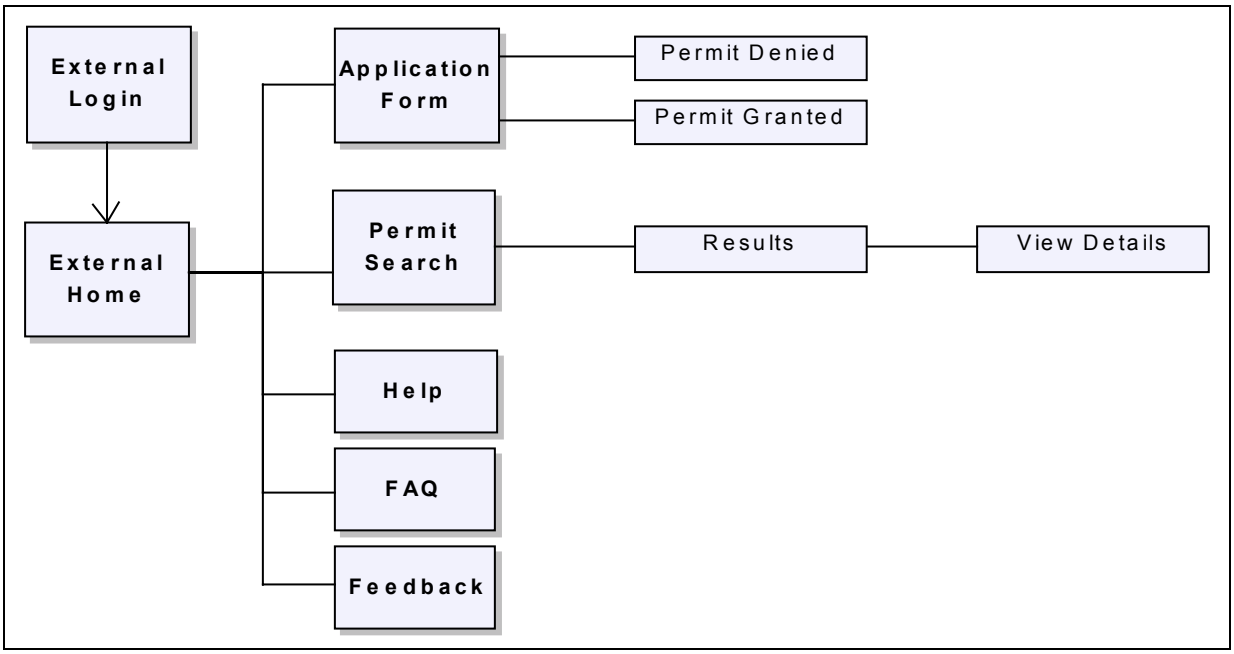


**Figure 10: External Web Pages Hierarchy**

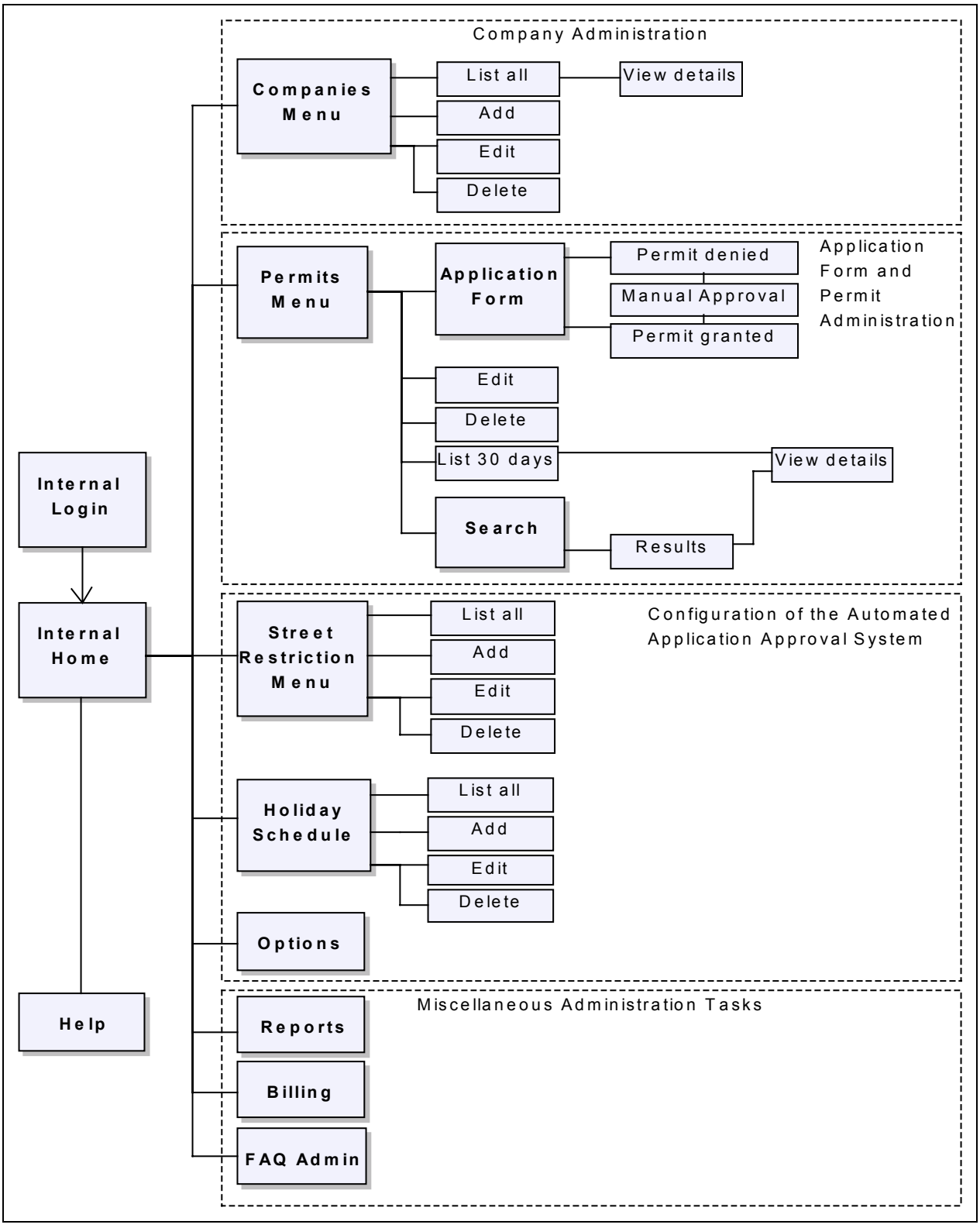


**Figure 11: Internal Web Pages Hierarchy**

### 4.2.2 Implementation using JSP

Since the goal was to use JSP only for generating formatted output and providing a user interface, particular attention was paid to placing the business logic in external classes (the business logic layer will be discussed in Section 4.3). All JSP pages are implemented in a common way. Therefore, just one example page is analyzed in detail below.

Presented here is the application form (file: *application_int.jsp*). The complete code for the page is not shown; repetitive code segments have been removed. JSP language segments are highlighted with a shaded background.

The beginning of the JSP page typically contains the HTML header. Also included here is the link to the CSS style sheet to achieve the uniform font and color style.

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="../style.css">
    <title>Permit System New Application</title>
  </head>
```

Then, the page language is set and the required Java classes are imported using the JSP *page* directive.

```
  <body>
  <%@ page language="java" import="java.sql.*,Permit.MyHTML,Permit.Company,
      Permit.WorkType,Permit.StreetWorkType,Permit.Street"%>
```

The JSP *include* directive is used to include the banner to the page.

```
  <%@ include file="../banner_permit.html" %>
```

The *NavigationBar* bean is instantiated, and the items that should be presented are added to the navigation bar content. Then, the completed navigation bar is dumped on the page with the evaluation of the *toHTML()* function.

```
  <jsp:useBean id="navApplicationInt" class="Permit.NavigationBar" />
  <% navApplicationInt.add("Home","home_int.jsp"); %>
  <% navApplicationInt.add("Permits Menu","permits_int.jsp"); %>
  <% navApplicationInt.add("Help","help_int.jsp"); %>
  <%= navApplicationInt.toHTML() %>
```

Another bean is invoked, the database bean *MyDbBean*. A connection the the Permit System database is established, and all information that is required for the dynamic content generation is read using static functions of the appropriate classes. The database connection is then released immediately with the *close()* command to free the connection for the next use.

```
<jsp:useBean id="dbBean" class="Permit.MyDbBean" />
<%
  dbBean.connect();
  ResultSet companies=Company.readAllCompanies(dbBean);
  ResultSet workTypes=WorkType.readAllWorkTypes(dbBean);
  ResultSet streetWorkTypes=StreetWorkType.readAllStreetWorkTypes(dbBean);
  ResultSet streets=Street.readAllStreets(dbBean);
  dbBean.close();
%>
```

Then, the page title is dislpayed using plain HTML code.

```
<table border=1 cellpadding=10 width=100%>
  <tr><td>
  <h2>File New Application</h2>
  <center>This form is to be used by the Department of Public Works only.
  </center>
  </td></tr>
</table>
```

The following code segment starts the main part of the page, which is the application form. The form is of encoding type *multipart/form-data* to allow for mixed content (a drawing file may be uploaded). This is also done using HTML only. The submitted form will be handled by a servlet (a servlet mapping has been defined in the web.xml file for *add.Permit*)

```
<form enctype="multipart/form-data" method="POST" action="add.Permit">
<table border=1 width=100%>
  <tr><td align=center bgcolor=#800000>
  <b><font color=#FFFFFF>Please provide the following information:
     </font></b>
  </td></tr>
</table>
<table>
```

The first item of the form is a drop-down box that is used to select a company name. The drop-down is generated automatically by the createDataSelect() function of the MyHTML class.

```
    <tr>
      <td bgcolor=#C0C0C0 align=right>Contractor/Utility Company:</td>
      <td>
        <%
          StringBuffer str=new StringBuffer();
          MyHTML.createDataSelect(str,"PerComName",1,companies);
        %>
        <%= str.toString() %>
      </td>
    </tr>
```

Other drop-down boxes for work types, dates, streets, etc. are created in a similar manner, as shown below. Form fields that do not contain dynamic data are created in HTML, e.g. the input field for the Dig Safe Number. Note: not the entire code for the form is shown here.

```
    <tr>
      <td bgcolor=#C0C0C0 align=right>Type of Work:</td>
      <td>
        <%
          str=new StringBuffer();
          MyHTML.createDataSelect(str,"PerWorkType",1,workTypes);
        %>
        <%= str.toString() %>
      </td>
    </tr>
    <tr>
      <td bgcolor=#C0C0C0 align=right>Dig Safe Number:</td>
      <td><input type=text name="PerDigSafe" size=30></td>
    </tr>
    <tr>
      <td bgcolor=#C0C0C0 align=right>First Day of Work:</td>
      <td>
        <%
          str = new StringBuffer();
          MyHTML.createDateSelect(str,"PerValidFromDay",
                                  "PerValidFromMonth","PerValidFromYear",3);
        %>
        <%= str.toString() %>
      </td>
    </tr>
```

The form has two buttons. The user may either submit or reset the entered application data.

```
  </table>
  <input type=submit value="Submit" name="Submit"> 
  <input type=reset  value="Reset"  name="Reset">
  </form>
```

Finally, the disclaimer is included to the page.

```
  <%@ include file="../disclaimer.html" %>
  </body>
</html>
```

The resulting output of the application form JSP page is presented in Figure 13 (on the next page).

The navigation between the Permit System pages was implemented through HTML hyperlinks. As an example, the Permit System's internal home page providing access to all internal function groups is shown in Figure 12. The implementation is trivial, and is not discussed in further detail here.


**Figure 12: Navigation: Internal Home Page Screenshot**

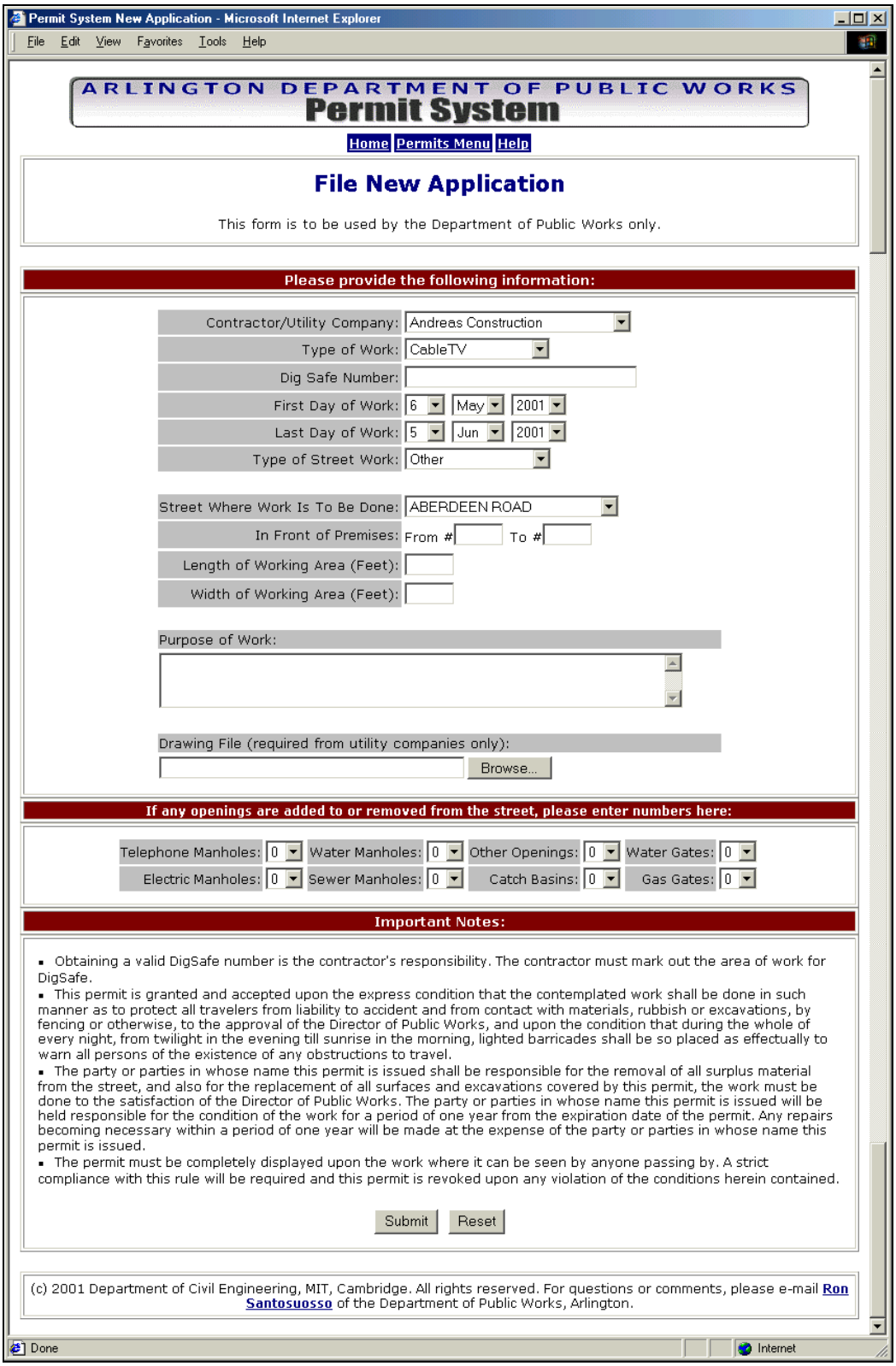


**Figure 13: Permit Application Form Screenshot**

## 4.3 Underlying Database in MySQL

MySQL is an Open Source database with a command line user interface. Prior to implementation of the Permit System data tables in MySQL, a graphical representation of the tables, attributes, and relationships was created to visualize the data. Although attributes had to be added to the tables during the development process, the overall concept and layout of the database remained the same. A graphical representation of the database is shown in Figure 14.

Figure 14 also illustrates the naming convention that was used for variables and database attributes. The first three letters of the attribute reflect the table name. Introducing this abbreviation as a part of the attribute name offered the advantage of not having to explicitly provide the table name in the SQL statements, since the attribute name is already the unique identifier.

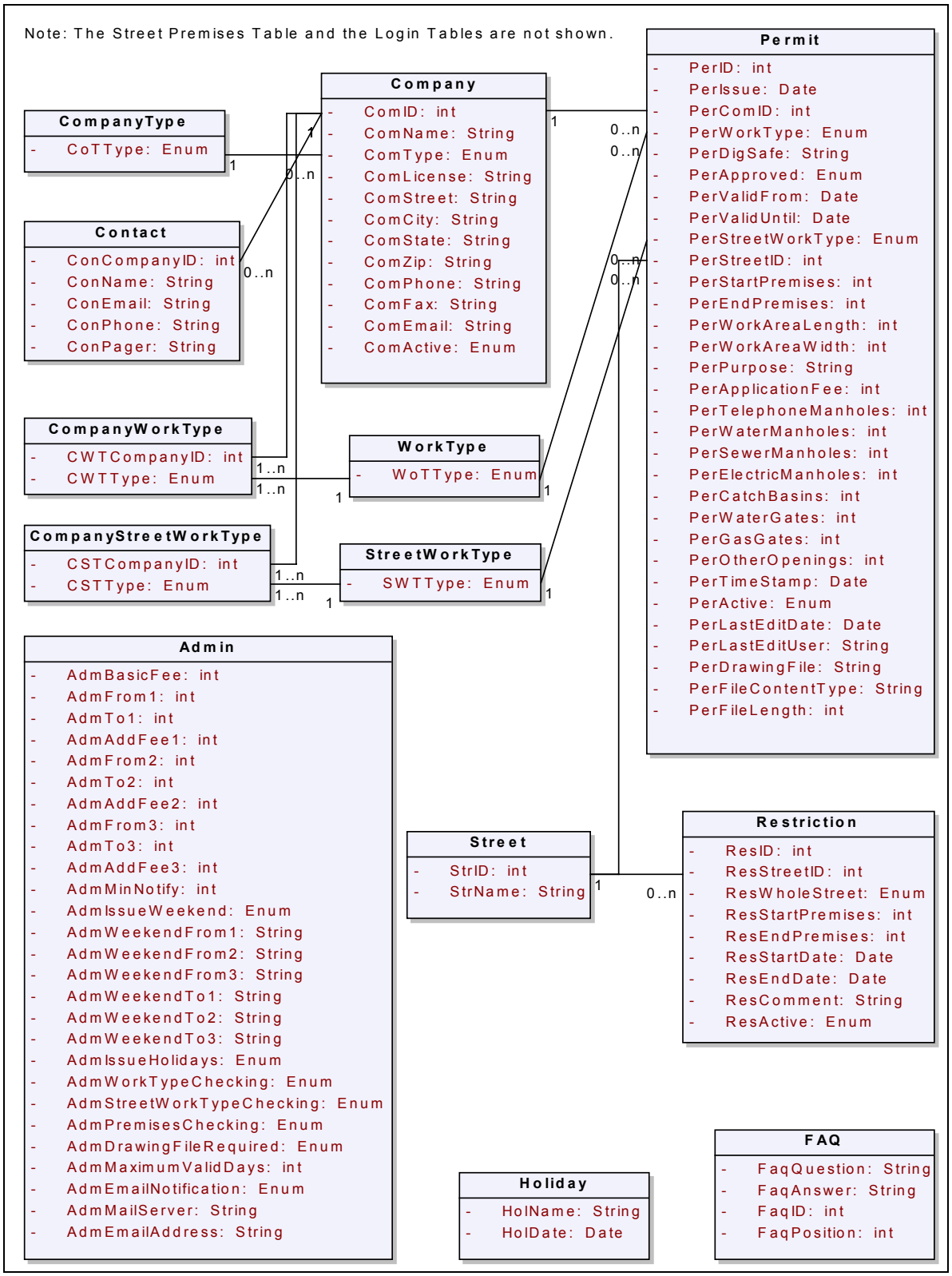


**Figure 14: Main Database Tables and Relationships**

The following table briefly describes the most important database tables and its attributes. The descriptions are provided to give the reader a general overview over the contents of the Permit System database. For a more detailed analysis of the tables and their attributes, please refer to the Street Opening Permit System Project Report that is available from the CEE Department of the Massachusetts Institute of Technology.

| Table Name | Primary Key(s) | Description |
|---|---|---|
| *Company* | *ComID* | The *Company* table contains the information on all contractors and utility companies that are registered with the Arlington Department of Public Works. |
| *Contact* | *ConCompanyID*<br>*ConName* | This table contains contact information for each company. Several contacts may be given. |
| *Company Type* | *CoTType* | This table contains the company types that are possible, which are *Utility* or *Contractor*. |
| *Company Work Type* | *CWTCompanyID*<br>*CWTType* | This table contains all types of work that a particular company may perform. E.g. a company registered for the type of work *CableTV* will not be allowed to perform work of type *Gas*. |
| *Company Street Work Type* | *CSTCompanyID*<br>*CSTType* | This table contains all types of street work that a particular company may perform. Similarly to the *Company Work Type* table, a company registered for street work of type *Street Opening Only* will not be permitted to perform work of type *Water*. |
| *Work Type* | *WoTType* | The *Work Type* table enumerates all possible work types. |
| *Street Work Type* | *SWTType* | The *Street Work Type* table enumerates all possible types of street work. |
| *Permit* | *PerID* | The *Permit* table contains a history of all issued permits, including all application data, such as issue date, location, size of the work area, etc. |
| *Street* | *StrID* | This table contains a list of all the streets that exist in the Town of Arlington. |
| *Restriction* | *ResID* | This table is used to store all restrictions that have been applied to street or street segments. Permits will not be issued for restricted streets. |

| Table Name | Primary Key(s) | Description |
|---|---|---|
| *Holiday* | *HolName* | This table contains the holiday schedule. Permits will not be issued on holidays. |
| *Admin* | - | The *Admin* table stores several parameters that are used to configure the behavior of the Automated Permit Approval System. This table does not have a primary key since it contains only one row. |
| *Frequently Asked Questions* | *FAQID* | This table contains a list of all frequently asked questions. The list of FAQ's is composed by the town's employees, and presented to external users. |

**Table 5: Description of Database Tables**

As mentioned earlier, MySQL provides a command-line user interface. On-line documentation explaining both basic and advanced features is available at

http://www.mysql.com/documentation/index.html.

The connection from the Web application to the database is usually done with a JDBC driver. The JDBC driver provides the middle tier from the Java SQL classes to the database. The JDBC driver serves to translate the Java SQL statement format to the database-specific format, and returns the results that the processing of the statement generated to the application.

The Permit System uses the MM MySQL JDBC driver, which was released under the GNU LGPL license. It is available at http://mmmysql.sourceforge.net/.

With the JDBC driver, connecting to the database becomes easy. The Permit System uses a simple JavaBean class to get a connection and to execute SQL statements. The connection is closed immediately after execution of the SQL statements. The *DbBean* class and its most important methods are presented below.

Due to the low expected load of the application (more than 5 concurrent users are very unlikely), no connection pooling was implemented (connecting to a database is a very resource intensive task. Heavy-duty applications therefore use connection pools that leave a set of connections open that are re-used many times. Please see Section 5.5.4 for further details on database connection pooling).

```
package Permit;
import java.io.*;
import java.sql.*;

public class DbBean {

  private Connection dbCon;
  private String dbURL=""; // a default database URL may be provided here.
  private String dbDriver="org.gjt.mm.mysql.Driver";

  // Set the database URL.
  public void setDbURL(String dbURL) {
    this.dbURL = dbURL;
  }

  // Open a database connection
  public boolean connect() throws ClassNotFoundException, SQLException {
    Class.forName(this.getDbDriver());
    dbCon = DriverManager.getConnection(this.getDbURL());
    return true;
  }

  //Close the database connection.
  public void close() throws SQLException {
    dbCon.close();
  }

  // Executes an SQL Query.
  public ResultSet execSQLQuery(String sql) throws SQLException {
    Statement s = dbCon.createStatement();
    ResultSet r = s.executeQuery(sql);
    return r;
  }
}
```

An example for a database query with the *DbBean* class looks like this. In the presented case, a list of all company names is extracted from the Company table.

```
// Instantiate the database JavaBean class
DbBean dbBean = new DbBean();
// Set the database URL
dbBean.setDbURL("jdbc:mysql://localhost/test_permit?user=&password=");
// Establish a connection
dbBean.connect();
// Execute the SQL query
ResultSet rs = dbBean.execSQLQuery("SELECT ComName FROM Company");
// Now, do something with the result set, e.g. display it!
// Don't forget to close the connection!
dbBean.close();
```

## 4.4 Implementation of the Application Logic

The JSP pages are the front-end of the application, while the database stores the system data. The application logic, programmed in Java, pulls the two pieces together. The following figure shows all classes and the dependencies among them for the internal part of the application.

Figure 15 indicates that many classes are servlet classes (i.e. they are derived from the *HTTPServlet* class). The servlet classes contain only *service()* methods to process client requests. The request is almost always received from a JSP page.

To the left of the servlet classes, the classes containing the application logic are shown. Their methods perform actions such as database queries, updates, and inserts according to provided arguments. Various *get()* methods are also provided, which are used by the JSP pages to access the dynamic data that the private member variables of the classes contain once a database query to retrieve the information was performed. The dependencies of the servlets to the application logic classes are shown in the diagram above through solid lines.

To the right of the servlet classes, miscellaneous supplementary classes are shown. Some of them are required by most of the servlets (e.g. the database bean class *MyDbBean*), while others are needed less frequently.

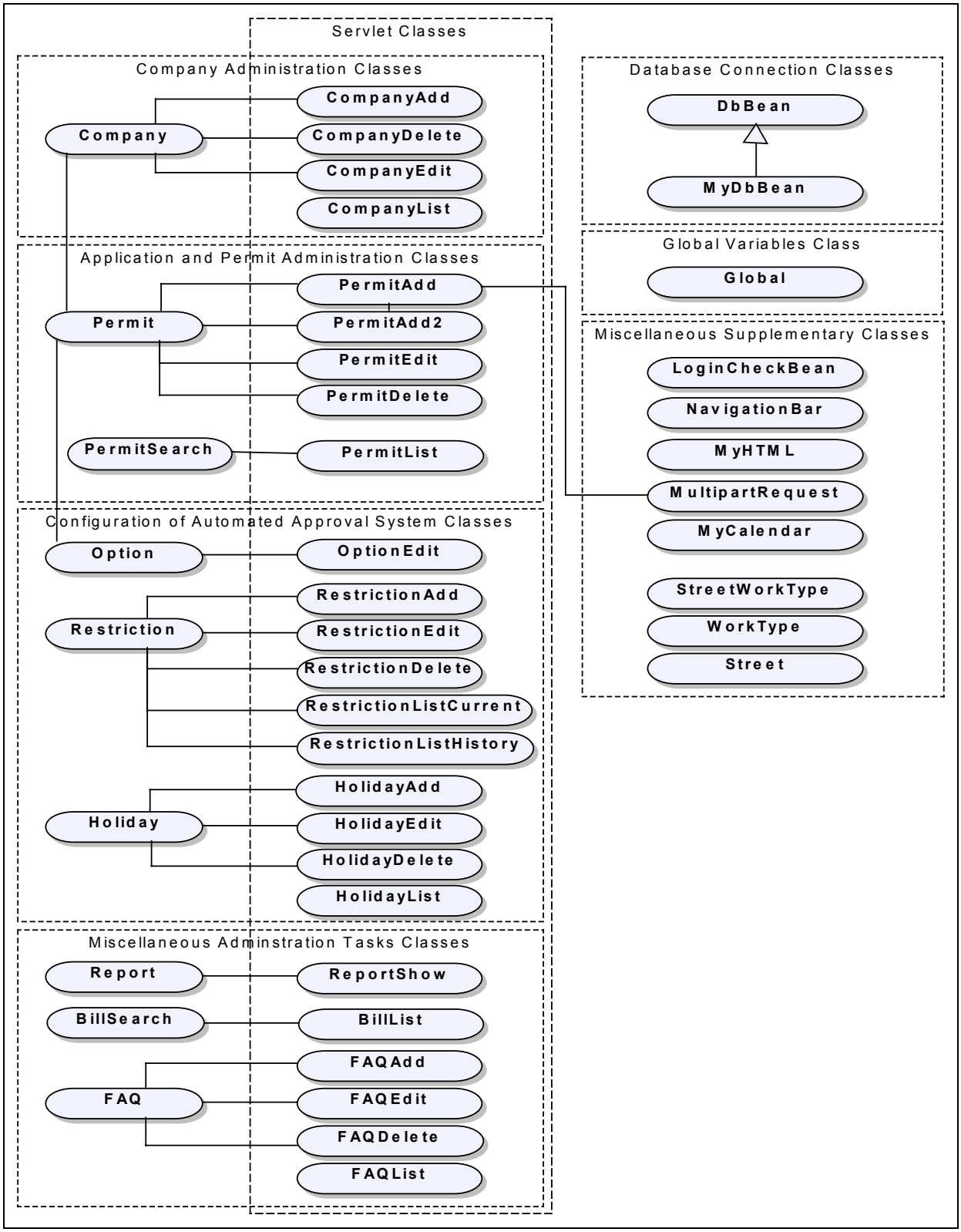


**Figure 15: Permit System Classes Overview**

The following table gives a brief description of all the internal Permit System classes.

| Class Name | Description |
|---|---|
| *Company* | This class contains the methods to access the company data in the database. Furthermore, contact information is processed with this class. |
| *CompanyAdd*<br>*CompanyEdit*<br>*CompanyDelete* | These servlets handle a user request for an addition, update, or deletion of a company. After completion of the task, a confirmation message is shown. |
| *CompanyList* | This servlet retrieves the list of all companies from the database. Using the *MyHTML* class, a formatted output of the list is presented to the user. |
| *Permit* | This class contains the methods to access the permit data in the database. The application form checking functions are also included to this class. |
| *PermitAdd* | This servlet processes a submitted application form using the checking functions of the *Permit* class. It utilizes the *MultipartRequest* class to upload drawing files that may have been submitted by the user. It generates an output providing the checking results, including options on how the user may continue (*Back to Application, Cancel, Proceed*). All application information is stored in the session object. |
| *PermitAdd2* | This servlet is reached if the application was approved by the system and confirmed by the user. The application data that was written to the session object in the *PermitAdd* servlet is retrieved, and new permit is stored in the database. |
| *PermitEdit*<br>*PermitDelete* | These servlets handle the requests for updating or deletion of permits. After completion of the task, a confirmation message is shown. |
| *PermitSearch* | This class contains variables and methods to extract specific permits from the database. |
| *PermitList* | This servlet utilizes the *PermitSearch* class to obtain a list of permits according to submitted search criteria or a list of all permits issued for the last 30 days. Using the *MyHTML* class, a formatted output of the list is presented to the user. |
| *Option* | This class contains variables and methods to retrieve or update the options data from the *Admin* table of the database. |
| *OptionEdit* | This servlet handles the update request for options. |
| *Restriction* | This class contains the methods to access street restriction data in the Permit System database. |

| Class Name | Description |
|---|---|
| *RestrictionAdd*<br>*RestrictionEdit*<br>*RestrictionDelete* | These servlets handle the requests for adding, updating, or deletion of street restrictions. A restriction may apply to a whole street or to a street section only. After completion of the task, a confirmation message is shown. |
| *RestrictionListCurrent*<br>*RestrictionListHistory* | These servlets retrieve the list of all current restrictions or the history of restrictions, respectively, from the database. Using the *MyHTML* class, formatted outputs of the lists are presented to the user. |
| *Holiday* | This class contains the methods to access holiday data in the database. |
| *HolidayAdd*<br>*HolidayEdit*<br>*HolidayDelete* | These servlets handle the requests for addition, updating, or deletion of holidays. After completion of the task, a confirmation message is shown. |
| *HolidayList* | This servlet retrieves the current holiday schedule from the database and displays it using the *MyHTML* class. |
| *Report* | This class contains the methods required to generate reports. |
| *ReportShow* | This servlet utilizes the *Report* class to display reports. |
| *BillSearch* | Closely modeled according to the *PermitSearch* class, this class is used to retrieve permit within a date range to generate billing information. |
| *BillList* | This servlet uses the *BillSearch* class to create a page in Excel spreadsheet format containing the billing information for the specified time period. |
| *FAQ* | This class contains the methods to access FAQ data in the database. |
| *FAQAdd*<br>*FAQEdit*<br>*FAQDelete* | These servlets handle the requests for addition, updating, or deletion of frequently asked questions. After completion of the task, a confirmation message is shown. |
| *FAQList* | This servlet retrieves all FAQ's from the database and displays them. |
| *DbBean* | This class provides the database connectivity. For a detailed description, please refer to Section 0 of this document. |
| *MyDbBean* | This class inherits from the *DbBean* class, and provides the Permit System-specific database URL. |
| *Global* | This class contains the application-wide constants. |
| *LoginCheckBean* | This class is used for basic user authentication. |
| *NavigationBar* | A navigation bar can be easily created with this class. |
| *MyHTML* | The *MyHTML* class contains many methods to generate HTML output from various sources, such as database result sets or arrays. Many output formats are possible. The most common ones are tables and drop-down boxes. |

| Class Name | Description |
|---|---|
| *MultipartRequest* | The *MultipartRequest* class is used to process the application form. Since the application form contains mixed content (both an input stream for a file upload, and regular form data), the processing is different from a standard *doGet()* servlet method. Please see Section 5.3 for a detailed discussion of file uploading with servlets. |
| *MyCalendar* | The *MyCalendar* class is the interface between the MySQL date format and the Java date format of the *GregorianCalendar* class. Conversion methods are provided in this class. |
| *Street*<br>*WorkType*<br>*StreetWorkType* | Supplementary classes to access the corresponding database tables. |

**Table 6: Permit System, Description of Internal Classes**

When a servlet is invoked by the servlet engine (i.e. the client has requested the servlet by selecting a hyperlink or pressing a button on one of the JSP pages), other methods from non-servlet classes are called by the servlet to perform the requested action. This could be storing information to, or retrieving information from the database, for example. The servlet also displays a result page, e.g. the confirmation that a database update was handled successfully, or the contents of a retrieved database result set. A typical servlet doing this is shown below.

```
package Permit;
import javax.servlet.*;
import javax.servlet.http.*;

import java.io.*;
import java.sql.*;

// Servlet class to list all companies.
public class CompanyList extends HttpServlet {
  // doGet() Handles the HTTP request for listing all companies.
  public void doGet(HttpServletRequest req, HttpServletResponse res)
           throws ServletException, IOException {

    // Instantiate a string buffer to hold the generated HTML page.
    StringBuffer str = new StringBuffer();
    MyHTML.setApplicationPath(Global.PERMITURL);

    // Start page output to the string buffer
    // Insert the page header including the style sheet link
    MyHTML.startPage(str,"List of All Companies","../style.css");
    // Add the page banner
    MyHTML.includeHTMLFile(str,"banner_permit.html");
    // Add the Navigation Bar
```

```
    NavigationBar naviBar = new NavigationBar();
    naviBar.add("Home","home_int.jsp");
    naviBar.add("Companies Menu","companies_int.jsp");
    naviBar.add("Help","help_int.jsp");
    str.append(naviBar.toHTML());

    // Display the page title
    MyHTML.insertTitle(str,"List of All Companies",2);

    // Instantiate a new database connection JavaBean
    MyDbBean dbBean= new MyDbBean();

    try {
      // Perform the database query to get the list of companies
      dbBean.connect();
      ResultSet rs = dbBean.execSQLQuery("SELECT ComID,ComName, "+
                     "ComLicense,ComActive FROM Company ORDER BY ComName");
      // Use the MyHTML class to generate a formatted table from the
      // result set
      String[] labels={"ComID","Company Name","License #","Active"};
       MyHTML.createHyperlinkDataTableWithID(
              str, labels, rs, 2,"ComID","company_details_int.jsp");
    }
    // Perform exception handling
    catch(SQLException e) {
      MyHTML.createDatabaseReport(str,MyHTML.ERROR,
             "A database error has occured.<br>"+e);
    }
    catch(ClassNotFoundException e) {
      MyHTML.createDatabaseReport(str,MyHTML.ERROR,
             "The database driver could not be found.<br>"+e);
    }
    finally {
      try {
        // close the database connection
        dbBean.close();
      }
      catch(SQLException e) {
        MyHTML.createDatabaseReport(str,MyHTML.ERROR,
               "The database connection could not be closed!<br>"+e);
      }
    }
    // Add the disclaimer
    MyHTML.includeHTMLFile(str,"disclaimer.html");
    MyHTML.endPage(str);

    // Get print writer of the servlet output stream
    PrintWriter out = res.getWriter();
    // Post the results by printing the string buffer to the output stream
    res.setContentType("text/html");
    out.println(str);
    out.close();
  }
}
```

A screenshot of the resulting output is shown below.

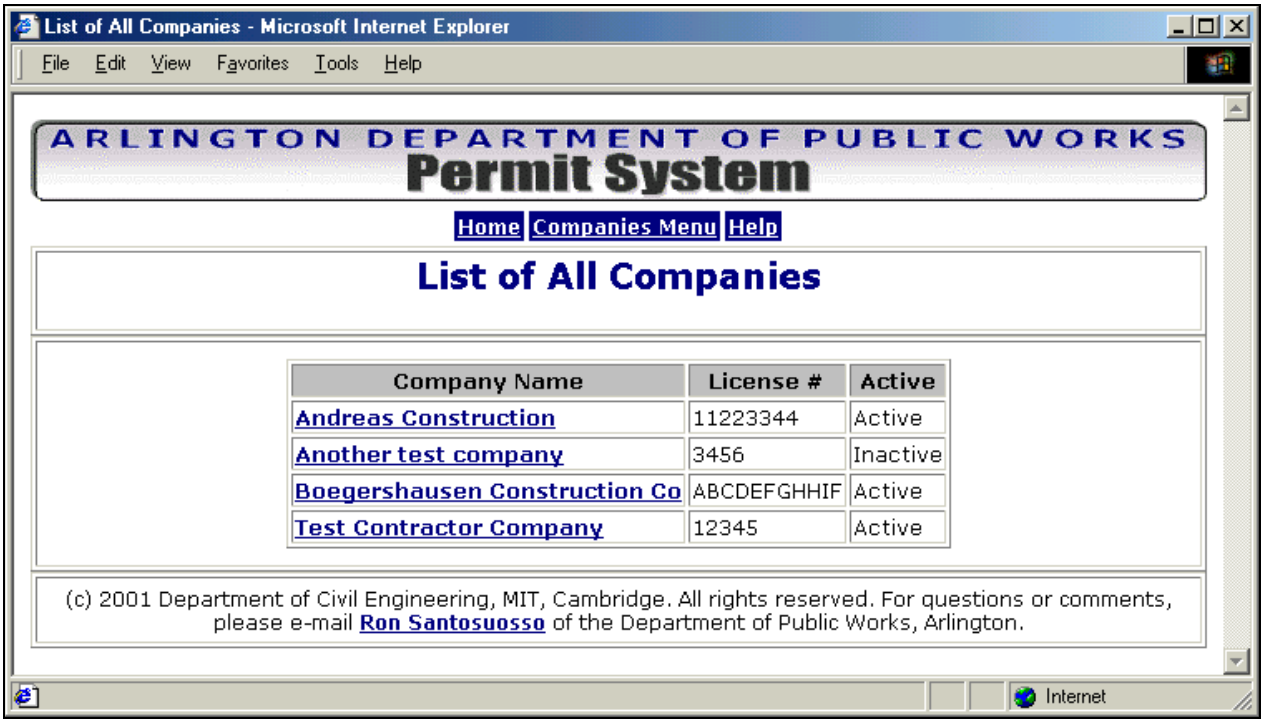


**Figure 16: Servlet Output Screenshot**

## 4.5 Source Code Organization

Figure 17 shows the organization of the source code for the Street Opening Permit System. It closely follows the suggestions provided by the Apache Software Foundation (see Section 3.2.1), and utilizes CVS for version management.

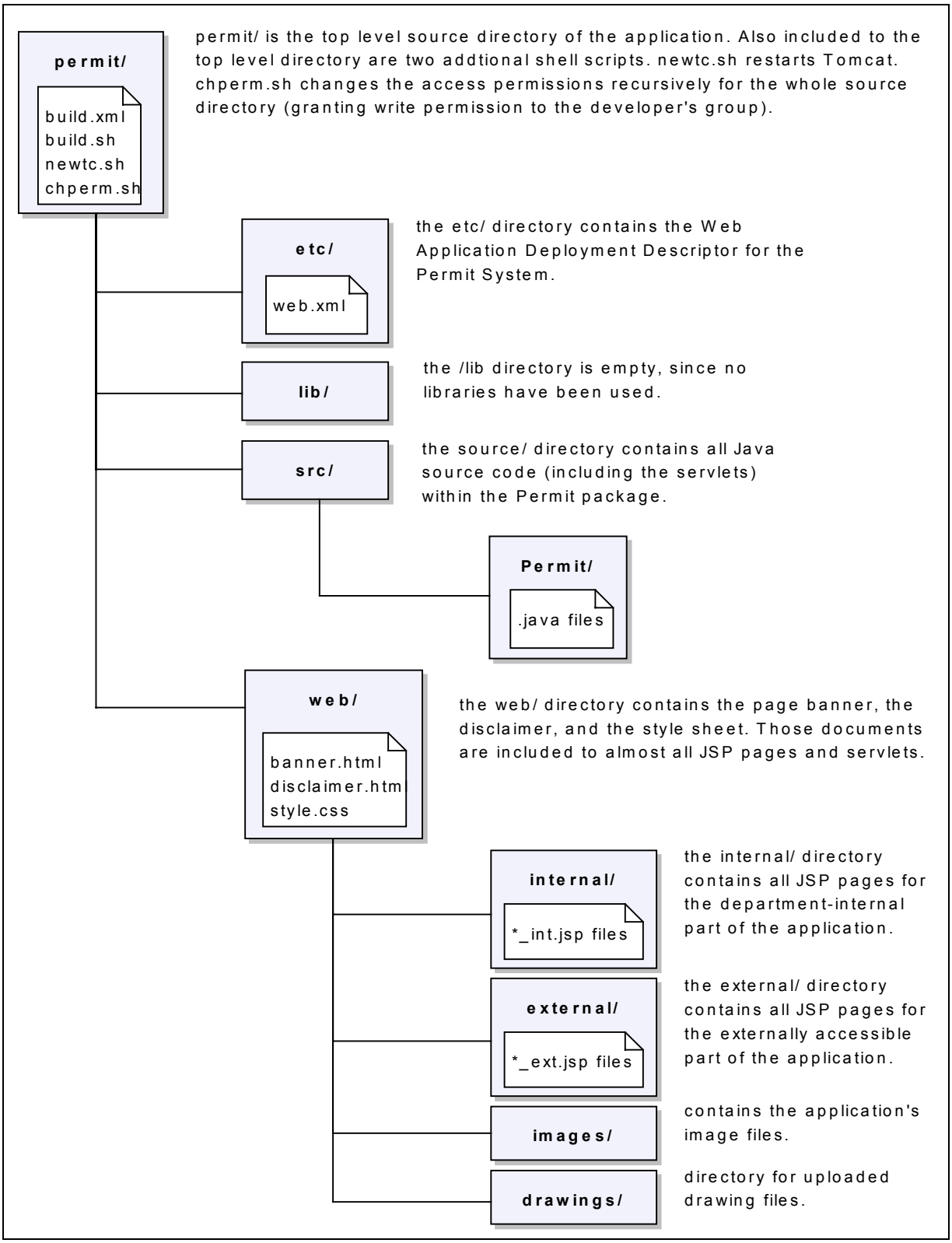


**Figure 17: Source Code Directory Structure for the Permit System**

## 4.6 Concluding the Case Study

### 4.6.1 Summary

The case study of the Street Opening Permit System for the Engineering Division of the Department of Public Works of the Town of Arlington has demonstrated the successful utilization of Open Source technologies for a real-world application.

A team of three students has concurrently worked on the implementation of the JSP pages, servlets, and other Java classes, mainly during the spring term of 2001. The project is currently about 90% complete, a few items are not fully implemented yet. The database will be integrated with the Pavement and Inspection Management System, another student project conducted at MIT this spring.

All Open Source components used (Red Hat Linux V7.0, MySQL V3.23, Tomcat V3.2.1, CVS) have been working reliably throughout the development and the testing phase of the software.

The complete source code of the Permit System has been put under the GNU GPL, and is available at the CEE Department of MIT.

### 4.6.2 Selected References


- Sun Microsystems; ***Servlet API Specification V2.2***; http://java.sun.com/products/servlet/2.2/javadoc/index.html
- Sun Microsystems; ***Java 2 Platform, Standard Edition, V1.3 API Specification***; http://java.sun.com/j2se/1.3/docs/api/index.html
- Web Design Group; ***Guide to Cascading Style Sheets***; http://www.htmlhelp.com/reference/css/
- Leon Changxin Qi; CEE Department of MIT; ***Enabling Technologies for a Web-based Urban Street Construction Permit System***; May 2001
- Rajesh Prasad; CEE Department of MIT; ***Pavement Permit System Infrastructure: UML Based Design***; May 2001
- Changxin Qi, Rajesh Prasad, W. Andreas Klimke; CEE Department of MIT; ***Street Opening Permit System Project Report***; May 2001

# 5 Miscellaneous Development Topics

## 5.1 Transferring Information between Linux and Windows Systems

Some developers may wish to use Windows for development, since they are more familiar with this environment, or since they already have easy access to a Windows system (e.g. their computer at home). Most Open Source products are available for Windows, and since Java, Servlets and JavaServer Pages are platform-independent technologies, there is no problem with developing on a Windows system while deploying the application on a Linux machine. The CVS repository, however, is recommended to be installed on a Linux system rather than on Windows, if multiple developers are involved.

This section introduces some of the options that are available in accessing the Linux server from a Windows machine, and vice versa.

### 5.1.1 Accessing the Linux Server over Telnet

Most Linux distributions include a telnet server package, which is often enabled by default. With a telnet server running on the Linux server, users can remotely login to the machine from any other machine over the telnet client application (a telnet client application is included to Microsoft Windows by default).

The telnet protocol server on Linux is called *telnetd* (for telnet daemon). Once the super-server application *xinetd* receives a request for a telnet connection, the telnet daemon is started automatically, and a connection to the remote machine is established. The user can then work on the server in a Linux box (if the username and password are accepted).

To check whether the telnet server is enabled, you may issue the following command to display the telnet configuration file. The *disable* parameter should be set to “no”.

```
$ more /etc/xinetd.d/telnet
```

```
service telnet {
        disable = no
        flags           = REUSE
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/sbin/in.telnetd
```

```
        log_on_failure  += USERID
}
```

To connect to the Server from the Windows machine, just run telnet (click “Start” from the Task Bar, then select “Run…”, and type *telnet {host name}*).

### 5.1.2 Transferring Files from and to the Server With FTP

A convenient way to transfer files to and from the Linux server is the File Transfer Protocol. The standard configuration for FTP is a private user-only site, which means that only users that have accounts on the server are allowed to transfer files, and files may only be transferred to and from the user’s home directory path. Since the developers should only work with local copies of the source code in their home directory, this FTP access mode is well suited.

As for the telnet server, an FTP server and an FTP client take care of the file transfer. The FTP server is invoked as the client requests an FTP connection.

You may connect to the Linux server either through the command line, or through a Web browser. To start the command line FTP client on the Windows machine, click “Start” from the Task Bar, then select “Run…” and type *ftp {host name}*. The ftp program will request a login username and password.

```
ftp theServer.mit.edu
```

From a browser, type *ftp://{username}@{hostname}* to connect. A window requesting the login password will be shown.

```
ftp://andreas@theServer.mit.edu
```

The command line tool provides more functionality than browser-based FTP. For example, files can be transferred in ASCII or binary mode (ASCII mode transfer takes care of the character format differences between Linux and Windows, and converts the character codes during transfer). Command line FTP also allows providing a script file that is run at the FTP session start time. A script file is useful to perform repetitive tasks.

The browser-based FTP is easier to use since it allows dragging and dropping of files in a graphical environment, but it is less flexible that the text-based FTP interface.

## 5.2 Multiple Developers under Tomcat

In this section, three different set-ups for the operation of Tomcat during development as a stand-alone Web server are introduced. Each of the approaches has its advantages and disadvantages. Depending on the number of developers and the number of machines available, the most appropriate solution should be selected. The techniques may be mixed as well.

### 5.2.1 One Instance of Tomcat on Multiple Servers

Advantages: No interference among developers when using Tomcat (starting, stopping, debugging). The full system resources are available to each developer.

Disadvantages: Tomcat must be installed and configured on all machines. Each developer must be familiar with configuring the *server.xml* file.

Giving each developer his/her Tomcat server is a good solution for advanced users that wish to have full control over the Web server, and are comfortable with configuring the Web Application Deployment Descriptor *web.xml* and Tomcat's *server.xml* file themselves.

A danger of distributing the deployment during development across many machines is that integration of the components may become more difficult, since it becomes more likely that the developers don't check on each other's work as much since they are not directly affected by it.

### 5.2.2 Multiple Instances of Tomcat on a Single Server

Advantages: No interference among developers when using Tomcat (starting, stopping, debugging). Direct traceability of errors to the originator is possible.

Disadvantages: System resources must be shared; difficult to configure; multiple *server.xml* files (one per developer).

With multiple instances of Tomcat, it is possible to provide each developer with his/her own Tomcat virtual machine. The application is then served at different port addresses. The configuration is quite complex. A separate *server.xml* file must be created for each user. Then, Tomcat needs to be started by providing the appropriate *server.xml* file.

Shown below are the *server.xml* configuration files for two developers. The differences are highlighted.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Server>
. . .
<Logger name="tc_log"
  path="/devteam/andreas/tc/tomcat.log"
/>
<Logger name="servlet_log"
  path="/devteam/andreas/tc/servlet.log"
/>

<Logger name="JASPER_LOG"
  path="/devteam/andreas/tc/jasper.log"
  verbosityLevel = "INFORMATION"
/>
<ContextManager debug="0"
  workDir="/devteam/andreas/tc/work"
  showDebugInfo="true" >
  <!-- ============= Interceptors ============= -->
  . . .
  <!-- ============= Connectors =============== -->
  . . .
  <!-- Normal HTTP -->
  <Connector

className="org.apache.tomcat.service.PoolTcpConnector">
   <Parameter name="handler"
   value=
   "org.apache.tomcat.service.http.HttpConnectionHandler"
   />
   <Parameter name="port" value="8080"/>
  </Connector>
  . . .
  <Context path="/myApp"
    docBase="/devteam/andreas/deployHome"
    debug="0"
    reloadable="true"
    trusted="false" >
  </Context>
</ContextManager>
</Server>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Server>
. . .
<Logger name="tc_log"
  path="/devteam/grace/tc/tomcat.log"
/>
<Logger name="servlet_log"
  path="/devteam/grace/tc/servlet.log"
/>

<Logger name="JASPER_LOG"
  path="/devteam/grace/tc/jasper.log"
  verbosityLevel = "INFORMATION"
/>
<ContextManager debug="0"
  workDir="/devteam/grace/tc/work"
  showDebugInfo="true" >
  <!-- ============= Interceptors ============= -->
  . . .
  <!-- ============= Connectors =============== -->
  . . .
  <!-- Normal HTTP -->
  <Connector

className="org.apache.tomcat.service.PoolTcpConnector">
   <Parameter name="handler"
   value=
   "org.apache.tomcat.service.http.HttpConnectionHandler"
   />
   <Parameter name="port" value="8081"/>
  </Connector>
  . . .
  <Context path="/myApp"
    docBase="/devteam/grace/deployHome"
    debug="0"
    reloadable="true"
    trusted="false" >
  </Context>
</ContextManager>
</Server>
```

The *server.xml* files should be placed in the users' home directories. To start the Tomcat processes, the following commands are used (the *–f* option indicates that the server configuration file is provided in the command line).

```
$ $TOMCAT_HOME/bin/startup.sh -f /devteam/andreas/tc/server.xml
$ $TOMCAT_HOME/bin/startup.sh -f /devteam/grace/tc/server.xml
```

### 5.2.3 One Instance of Tomcat on a Single Server

Advantages: Easy to configure (only single *server.xml* file).

Disadvantages: Interference among developers when using Tomcat (starting, stopping, debugging). Source of error messages is identifiable, but more effort is needed. The system resources must be shared among all developers.

Installing just one instance of Tomcat on a single server is the quickest way of setting up the environment. This is often sufficient for smaller development teams (for the case study presented in Section 4, this worked well for a group of three people; the team shared the server for testing and compilation of the application, while editing of the code was mostly done on other machines. In addition to the group of three, the server was occasionally used by two other development teams at the same time – however, up to 5 concurrent developers did not pose any problem).

To have multiple teams or people working under the same Tomcat process, all you need to do is add new contexts to the *server.xml* file (see example below). As long as Tomcat does not need to be restarted often (which is required to apply changes to the *web.xml* file), this can work well during development. By default, Tomcat allows for up to 50 concurrent threads, which is more than sufficient during development even with many team members.

```
        <Context path="/andreas"
                 docBase="/devteam/andreas/deployHome"
                 debug="0"
                 reloadable="true"
                 trusted="false" >
        </Context>
        <Context path="/grace"
                 docBase="/devteam/grace/deployHome"
                 debug="0"
                 reloadable="true"
                 trusted="false" >
        </Context>
```

**Reference**


- Gal Sachor; ***Tomcat - A Minimalistic User's Guide***; 1999
  http://jakarta.apache.org/tomcat/jakarta-tomcat/src/doc/uguide/tomcat_ug.html

## 5.3 Handling Mixed Content using a Servlet Input Stream

Forms sometimes require the uploading of files in addition to regular parameter data. An example is the application form of the case study presented in Section 4. In addition to entering form fields, the user has the option to submit a drawing file.

Instead of a simple “post” request, which is usually used to submit form data, the CGI “multipart/form-data” encoding type has to be used to allow uploading of files through a stream.

Unfortunately, the servlet API V2.2 does not support multipart/form-data requests very well. The input stream has to be parsed manually for the parameter value pairs and for the files contained in the stream. It turns out that this is quite complex to program – therefore, it makes sense to use an existing class library to perform this task.

A couple of libraries exist, but not all of them are free to use. For the case study presented in Section 4, Jason Pell’s *MultipartRequest* class was used, which was published under the GPL (the latest version is published under the LGPL). The *MulitpartRequest* class is available for download at http://www.geocities.com/jasonpell/programs.html, including source code and documentation. With this class, the multipart/form-data request can be handled in a very similar way to a regular post request. A simple example consisting a HTML form and the corresponding servlet is given below. Please make sure that the upload directory exists.

This is the HTML form.

```
<html>
 <head>
   <title>MultipartRequest Demo</title>
 </head>
 <body>
   <h2>MultipartRequest Demo</h2>
   <form enctype="multipart/form-data" method="POST" action="upload.Servlet">
     Please enter your name:
       <input type=text name="Name" size=30><br>
     File to upload:
       <input type=file name="File" size=30><br>
     <input type=submit value="Submit" name="Submit">
   </form>
 </body>
</html>
```

And this is the servlet handling the multipart/form-data request. Don't forget to define a servlet mapping in the *web.xml* file.

```
import javax.servlet.*;
import javax.servlet.http.*;

import java.io.*;
import java.sql.*;

public class UploadExample extends HttpServlet
{
  public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

    // Get the content type and the content length
    String contentType = req.getContentType();
    int contentLength = req.getContentLength();

    // Get the servlet input stream
    ServletInputStream in = req.getInputStream();

    // Perform the multipart request. This stores the file to the specified
    // directory, and generates a hashtable containing all the parameters
    // that are submitted with the form.
    // The paramters can be accessed with the getURLParameter(String) method.
    MultipartRequest multipartRequest =
      new MultipartRequest(contentType,contentLength,in,"/uploadDir");

    // read form paramters
    String name = multipartRequest.getURLParameter("Name");

    // get the file
    File file=multipartRequest.getFile("File");

    // Get print writer
    PrintWriter out = res.getWriter();
    // Print out a table of all items read
    out.println(multipartRequest.getHtmlTable());
    out.close();
  }
}
```

## 5.4 Documenting Source Code with the Javadoc Utility

A task that is often not paid enough attention to is the documentation of the source code, although this is crucial to achieve high maintainability of the product, as well as reusability of components.

Furthermore, in a project involving multiple developers, the developers have to interface with classes that other members of the team have written. Most likely, they will also want to use the methods of those classes. It is obvious that this is only possible if the classes are documented well, and the documentation is made available to the team members not at the end, but as early as possible in the development process.

A great tool to document Java source code is the *Javadoc* utility. Javadoc generates documentation in HTML format, including package trees, an index, and sophisticated navigation and cross-references. Since Sun Microsystems uses the same tool to generate their API reference documentation, the interface is well known to the Java developer community, and therefore easily understood.

The Apache Software Foundation has already provided the basis to use Javadoc in their *build.xml* file for application development with Tomcat. To automatically generate the HTML documentation for the Web application, there are only a few steps required, which are outlined below.

Step1: When writing your source code, comment it according to the format specified by the Javadoc documentation (see references). Basically, documentation text is inserted prior to each class and method declaration. Tags starting with an `@` symbol have a special meaning and are interpreted by the Javadoc compiler (please refer to the Javadoc documentation for a complete list of available tags).

```
/**
 * Servlet to demonstrate the MultipartRequest class by Jason Pell.
 * @author  W.Andreas Klimke (MIT)
 * @see MultipartRequest
 */
public class UploadExample extends HttpServlet {
 /**
  * Uploads a file to the /uploadDir directory.
  * @param req the request object
  * @param res the response object
  * @throws ServletException
  * @throws IOException
```

```
   */
   public void doPost(HttpServletRequest req, HttpServletResponse res)
     throws ServletException, IOException {
     . . .
   }
}
```

Step2: Generate an HTML file called *package.html* for each source code package, and place it in the package directory. This is important! The Javadoc utility looks for this file in each package folder.

```
<html>
  <head>
    <tile>Documentation of the example package</TITLE>
  </head>
  <body>
    Included are all Java classes of the example package.
  </body>
</html>
```

Step 3: Modify the *build.xml* file regarding the Javadoc command option, as shown below. The package names (which are as well the directory names) should be given under the *packagenames* parameter.

```
  <target name="javadoc" depends="prepare">
    <javadoc packagenames="myPackage"
             sourcepath="src"
             author="true"
             version="true"
             bottom="Copyright (c) 2001 MIT. All Rights Reserved."
             destdir="${javadoc.home}"/>
  </target>
```

Step 4: Build the documentation with the following command:

```
$ ./build.sh javadoc
```

```
javadoc:
  [javadoc] Generating Javadoc
  [javadoc] Javadoc execution
  [javadoc] Loading source files for package myPackage...
  [javadoc] Constructing Javadoc information...
  [javadoc] Building tree for all the packages and classes...
  [javadoc] Building index for all the packages and classes...
  [javadoc] Building index for all classes...

BUILD SUCCESSFUL
Total time: 3 seconds
```

Javadoc will generate the documentation under the specified destination directory. The generated files can be viewed through a browser without any additional modification, as shown in Figure 18.

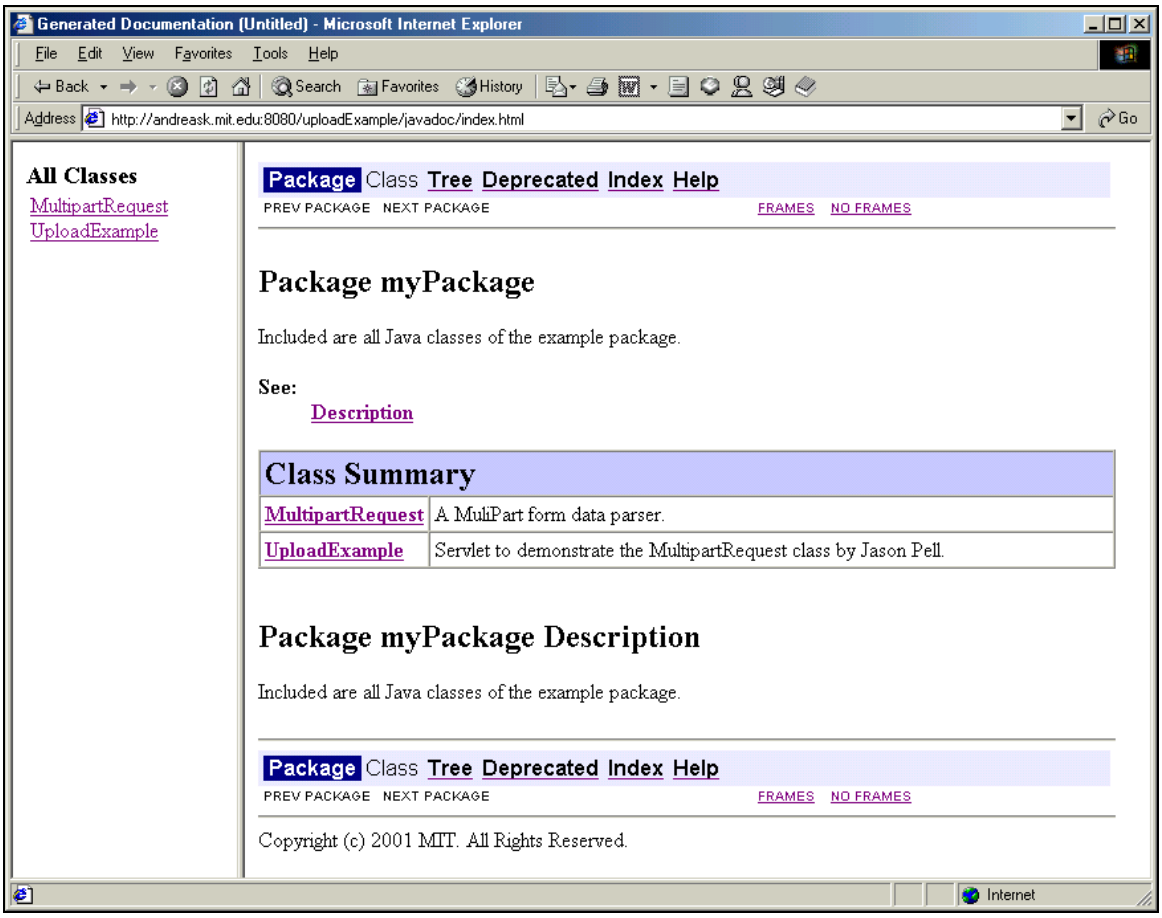


**Figure 18: Screenshot of Generated Documentation**

## References


- Sun Microsystems; ***How to Write Doc Comments for the Javadoc Tool***; http://java.sun.com/j2se/javadoc/writingdoccomments/

- Sun Microsystems***; Javadoc 1.3 Documentation***; http://java.sun.com/j2se/1.3/docs/tooldocs/javadoc/index.html

## 5.5 Increasing the Application's Performance

### 5.5.1 Good Programming Practices

Following is a list of good programming practices to increase performance. These practices are not just specifically for web application development. However, due to the resource-intensive nature of servlets, good code becomes even more important. (A servlet is instantiated once, and then executed many times, depending on the load of the web site. Inefficiencies can be magnified over time).

- Avoid string concatenation. Java concatenates two strings internally by instantiating a StringBuffer object, and using the StringBuffer's *append()* method to append the second string to the first string. When concatenating several strings with the + operator, several instances of string buffer objects are created, used, and then garbage collected, thus, system resources are wasted. String concatenation can be avoided by using the StringBuffer object as shown in the code example.

```
// StringBuffer example
String user = "John";
// The pre-defined size of the StringBuffer is 16 characters.
// To avoid the need for resizing, set the initial size to 50 chars.
StringBuffer strBuffer = new StringBuffer(50);
strBuffer.append("The user ");
strBuffer.append(user);
strBuffer.append(" is currently logged in.");
out.println(strBuffer.toString());
```

- Avoid unnecessary object instantiations. Each object that is instantiated requires system resources (memory) during its life cycle. Try to avoid the creation of new objects if possible.
- Free resources when they are no longer needed. When an object is no longer needed, it can be marked to be garbage collected. In Java, simply set the object to null, and the garbage collector can free the associated resources. (This is not required for local variables, since they automatically go out of scope when a function is terminated.)
- Use final classes and methods. The compiler will optimize a class declared final. This is because the compiler knows no subclasses can ever be created in the hierarchy. Similarly, a final method can be inlined by the compiler to avoid the overhead of an extra function call.

- Use static methods where applicable. A method declared static tends to execute faster than any other type of method declaration. Next are final methods, followed by instance methods, and last, synchronized methods.

### 5.5.2 Apache and Tomcat Web Server Integration

The Tomcat web server can serve both static and dynamic web pages. However, if a web site contains a significant number of static pages, using the Apache web server for the static content can increase the application's performance. The Apache web server is also an open source product developed by the Jakarta project (like Tomcat).

Besides the performance increase, there are several other reasons that make the combination of Apache with Tomcat a good choice:

- The Tomcat web server is not as configurable as Apache.
- Tomcat is not as robust as Apache.
- Apache provides some additional functionality that cannot be found under Tomcat (e.g. modules for Perl or PHP).

When running in combination with Apache, Tomcat is configured as an add-on to the Apache web server. Apache and Tomcat work together in the following way: Apache serves as the main web server that listens for client requests. Before Apache processes a request, the request is checked whether it refers to a dynamic web page (a servlet or JSP page). If a servlet or JSP page is requested, Apache forwards the request to the Tomcat web server, which processes it. Otherwise, Apache directly serves the request.

To achieve this behavior, both Apache's and Tomcat's configuration files have to be modified. Furthermore, an additional module has to be added to the Apache environment (either mod_jserv or mod_jk – the latter is the newer module, it is easier to configure and can handle the secure https protocol).

**Selected References:**

- Gal Shachor; ***Tomcat-Apache HOWTO***; http://jakarta.apache.org/tomcat/jakarta-tomcat/src/doc/tomcat-apache-howto.html; 1999-2000; Introduces the cooperation of Tomcat and Apache. Shows how to configure Apache and Tomcat. Also covers configuring Apache and Tomcat for multiple Tomcat JVM's, virtual hosting, and troubleshooting.

- Gal Shachor; ***Working with mod_jk***; http://jakarta.apache.org/tomcat/jakarta-tomcat/src/doc/mod_jk-howto.html; 1999-2000; Explains the installation and configuration of mod_jk more in-depth than the article mentioned above.

### 5.5.3 Prepared SQL Statements

When the same SQL statement needs to be executed repeatedly, it is more efficient to use the *PreparedStatement* rather than the *Statement* command. Prepared statements are precompiled prior to being used, and accept any number of parameters. A simple example of how to use prepared statements in Java is given below. The *PreparedStatement* class is part of the SQL Package of the core Java 2 API.

```
// Generate the SQL string
// To indicate a parameter, a question mark is used.
StringBuffer sql = new StringBuffer(256);
sql.append("INSERT INTO Users(Username, Password) ");
sql.append("VALUES(?, ?)");
// Create the prepared statement; assuming dbConn represents an existing
// database connection.
PreparedStatement stmt = dbConn.prepareStatement(sql.toString());
// Generate some random data to be put in the database
String[][] user = {
   { "johnt", "youllneverguessit" },
   { "garys", "topsecret" },
   { "eddym", "mybirthday" }};
// Execute the statement 3 times, after setting the parameters.
for (int i = 0; i < 3; i++)
{
   stmt.clearParameters();
   for (int j = 0; j < 2; j++) stmt.setString(j+1, user[i][j]);
   stmt.executeUpdate();
}
```

### 5.5.4 Database Connection Pools

The classical way of accessing a database is a three-step procedure:

(1) The database driver establishes a connection
(2) An SQL statement is executed, and a result set is retrieved
(3) The database driver closes the connection.

With this procedure, a new database connection object is created with each new client request. This is very time consuming, since the database engine must allocate communication and memory resources, authenticate the user, and set up a security context (this can easily take one or two seconds). Therefore, the performance of database access can be significantly improved by avoiding the opening of new connections with each client request. This is done by implementing database connection pools.

The concept of database connection pooling can be described as follows:

- An application gets a reference to the connection pool, or an object managing many pools (depending on the complexity of an application, it may be necessary to implement a pool manager that maintains multiple instances of connection pools).
- The application obtains a connection object from the connection pool. The connection pool maintains a collection of many database connections, which are kept in an open state to save the time that is required to establish a connection.
- The connection is used, i.e. a database statement is executed with the connection object.
- The connection is returned to the pool, and is therefore free for the next use.

Connection pools benefit most server applications. Performance can be improved significantly if the following conditions are satisfied (according to *Professional Java Server Programming*):

- Users access the database through a limited set of generic database user accounts, as opposed to a specific account per user.
- A database connection is only used for the duration of a single request, as opposed to the combined duration of multiple requests from the same client.

There are plenty of connection pool implementations described in computer literature. The book *Professional Java Server Programming* presents three different implementations of varying complexity to cover basic as well as advanced needs for connection pooling. Enterprise JavaBeans also offer database connection pooling.

# 6 Comparison of JSP and Java with ASP and C#

An alternative to Open Source Web application technologies, which predominantly use Java as the programming language, is available with Microsoft's Web servers, namely Personal Web Server (PWS) and Internet Information Server (IIS). Microsoft's Web application enabling technologies are Active Server Pages (ASP), which will soon enter a new stage with the introduction of the .NET platform and C#. In this chapter, the author takes a closer look at ASP and C# as the core technologies of Microsoft's Web application development platform, and compares them to Sun Microsystems' Java products.

## 6.1 JavaServer Pages versus MS Active Server Pages

The main purpose of both JSP and ASP is to add dynamic capabilities to web sites. This is done on the server side, i.e. the browser request is processed on the server before a dynamically generated web page is sent to the client. This extends the functionality of web applications significantly beyond the initial approach of generating dynamic content directly on the client (e.g. with client-side scripting languages or Java Applets).

JavaServer Pages and Java Servlets have been introduced in detail in earlier chapters. In this section, Microsoft's ASP 3.0 and the announced ASP.NET will be introduced briefly, and then compared to Servlets and JSP.

### 6.1.1 Microsoft Active Server Pages 3.0

An Active Server Page combines HTML, scripting, and server-side components in one file called Active Server Page. Supported scripting languages are VBScript and JavaScript (other scripting languages are supported through third parties).

ASP provides the programmer with a few objects that can be accessed from the ASP page without explicit instantiation. The objects have application, session, or page scope, and are used to process client interactions and store any data that is required to manage these interactions. The available objects are listed below.

- The Application object is used to share information among all users of a given application.
- The ObjectContext object is used to either commit or abort a transaction (managed by the Microsoft Transaction Server) that has been initiated by a script contained in an ASP page.
- The Request object retrieves the values that the client browser passed to the server during an HTTP request, and makes these values easily accessible to the programmer.
- The Response object is used to send output to the client.
- The Server object provides access to methods and properties on the server. Most of these methods and properties serve as utility functions.
- The Session object is used to store information needed for a particular client session. Variables stored in the Session object are not discarded when the user jumps between pages in the application; instead, these variables persist for the entire user-session. Session state is only maintained for browsers that support cookies. Note: ASP 3.0 does not support URL rewriting to maintain session state.

An important aspect of any Web application is the integration with databases. Active Server Pages support this well. The following code example illustrates the ActiveX Data Object controls for database connectivity. Since IIS Version 4.0, automatic connection pooling is available, so the simple method of opening and closing connections can even be used for larger scale applications.

```
<%@ language="VBScript" %>
<HTML>
  <HEAD><TITLE>Dispay user names</TITLE></HEAD>
<BODY>
<H2>User names in data base:</H2>
<% Dim Conn
   Dim UserNames
   ' Create a database connection object
   set Conn = Server.CreateObject("ADODB.Connection")
   ' Create a result set object
   set UserNames = server.CreateObject("ADODB.RecordSet")
   ' Establish the database connection (providing no username or password)
   Conn.Open "Provider=Microsoft.Jet.OLEDB.4.0; Data Source = " & _
     "C:\dataBases\Users.mdb"
   ' Execute the SQL statement
   Set UserNames = Conn.Execute("SELECT UserNames FROM Users")
   ' Display the results
```

```
   Do Until UserNames.EOF
      Response.Write UserNames("UserName<br>")
      RSUserNames.MoveNext
   Loop
   ' Close the database connection
   Conn.Close
%>
</BODY>
</HTML>
```

Scripting languages, as used by ASP, were thought of as true RAD languages, since they are very simple to use (e.g. no strong variable types are required). However, with the increasing complexity of Web applications, the usage of scripting proved to incur some serious limitations. Exception handling, for example, is possible in ASP, but it is very cumbersome as the following code excerpt demonstrates. The scalability of VB-style error handling is poor, since VBScript or Jscript do not provide the means to process errors with throwing and catching.

```
<%@ language="VBScript" %>
<%
Sub Foo()
   . . .
   On Error Goto ErrHandler
   Set Conn = Server.CreateObject("ADODB.Connection")
   . . . ' Some SQL Statements
   Exit Sub
   ErrHandler:
      Call ErrHandler(Conn)
End Sub

Sub ErrHander(Conn)
   NumberOfErrors=Conn.errors.count
   for Counter = 0 to NumberOfErrors-1
      Number=Conn.errors(counter).number
      Description=Conn.errors(counter).description
      Response.write "Error=" & Number & ", " & Description;
   Next
   . . . ' Also close any existing connections
End Sub
%>
```

Another problem with scripting languages is that an organization of the application into classes and libraries (namespaces) is not possible. This decreases maintainability and clarity especially in larger projects. Reuse of code in ASP 3.0 is difficult due to the lack of namespaces. To achieve reuse in ASP 3.0, functions need to be maintained in separate asp files. Then, these files have to be added to each page with an include directive. Another way of reuse is the generation of COM's, but this requires additional development time and effort.

### 6.1.2 Microsoft ASP.NET

ASP.NET will be part of Microsoft's .NET framework, and is currently only available as a beta version (March 2001). It has been referred to earlier as ASP+, but the name has been slightly changed by Microsoft to indicate its ties to the .NET platform.

The major improvement of ASP.NET over ASP is the introduction of C# as the server-side scripting language. Optionally, other third generation languages may be used for scripting. This will solve the problems of ASP 3.0 that were described above. Other improvements include a new object model for HTML controls called "server controls", which make the generation of HTML controls such as selection boxes, lists, or inputs much easier and provide a convenient way of separating the dynamic contents from the HTML code (Please see the code example below).

```
script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs e)
   {
      . . .
      DataSet dataset = . . . // (get data set from database)
      // Bind the data set DataSet to the DataList Control
      myDataList.DataSource = dataset.Tables[0].DefaultView;
      myDataList.DataBind();
      . . .
   }
</script>
<p>
   Please select the desired item:
   // This will be replaced by HTML code once the page is processed & sent
   // to the client. The runat attribute specifies that it is processed at
   // the server.
   <asp:DataList id="myDataList" runat="server">
</p>
```

Due to ASP.NET's close ties to Microsoft servers, some features such as page caching, database connection pooling, and authentication services (among others) are easily available to the programmer without having to implement these functionalities him/herself.

### 6.1.3 Comparison of JSP, ASP, and ASP.NET

Although ASP and JSP seem similar at a first glance, there are quite significant differences in the architectural approach of the two technologies. ASP.NET picks up some of the concepts that have made JSP successful, and addresses problems that developers faced with ASP 3.0. The following table summarizes the features of the technologies.

| Feature | JavaServer Pages | Active Server Pages 3.0 | ASP.NET |
|---|---|---|---|
| Supported web servers | Any web server (e.g. Apache, Netscape, IIS) | Microsoft IIS or Personal Web Server | Microsoft IIS (initially) |
| Supported platforms | Microsoft Windows, Solaris, Mac OS, Linux, and other Unix Platforms | Microsoft Windows (and others with 3rd party products) | Microsoft Windows |
| Extensible tags/ Extensibility Model | Create custom tag libraries | Custom tags only with IE 5 and XSL transformation | Server controls object model that can be extended |
| Reusable components | Supported across platforms | Reusable within the Windows platform, but cumbersome | Supported within the Windows platform |
| Scripting languages | Java, including the complete Java API | JavaScript, VBScript (others through 3rd parties) | C# or any other .NET-compliant language (initially VB and C++) |
| Integration with data sources | JDBC/ODBC drivers provided by the database vendors and JDBC API | ODBC/OLEDB drivers and ADO object | ODBC/OLEDB drivers and ADO.NET object |
| External components | JavaBeans and Enterprise JavaBeans | Through COM | Backward compatibility with COM, .NET/C# components |
| Extensive tool support | Yes | Yes | Expected |

**Table 7: Comparison of JavaServer Pages and Microsoft ASP**

The main advantages of JavaServer Pages over ASP 3.0 are the platform independence, and the benefits gained from enabling the server pages with a fully object-oriented programming language (this is especially valuable for enterprise applications, where clear structuring, maintainability, and appropriate error handling become important factors). ASP.NET promises to improve ASP 3.0 significantly in these areas. Developers have complained about the COM objects being inconvenient and cumbersome to use in ASP 3.0. With C# and ASP.NET, COM objects will still be supported, but no longer required as the only means to create components.

### 6.1.4 Selected References


- Ron Wodanski, Microsoft; ***ASP Technology Feature Overview***;
  http://msdn.microsoft.com/workshop/server/asp/aspfeat.asp; August 1998

- Len Cardinal, George V. Reilly, Microsoft; ***25+ ASP Tips to Improve Performance and Style***;
  http://msdn.microsoft.com/workshop/server/asp/ASPtips.asp; April 2000
  This article presents tips for optimizing ASP applications and VBScript. Includes a long list of additional references.

- J.D. Meier, ***Microsoft; Leveraging ASP in IIS 5.0***;
  http://msdn.microsoft.com/workshop/server/asp/server02282000.asp; February 2000
  Describes new features & improvements of ASP that where introduced in IIS 5.0.

- Chris Kinsman; ***Introduction to ASP+***;
  http://www.asp-zone.com/articles/ck072600/ck072600.asp; July 2000
  Summarizes the limitations and problems of ASP 3.0 from a developer's standpoint. Introduces the new features of ASP+ that solve these problems, such as compiled language integration (e.g. C#) instead of the scripting languages (JScript and VBScript), new page framework & object model, deployment, authentication, and load balancing architecture.

- Rob Howard, Microsoft; ***Five Steps to Getting Started with ASP.NET***;
  http://msdn.microsoft.com/library/default.asp?URL=/library/welcome/dsmsdn/asp11122000.htm;
  November 2000.
  Download instructions of the .NET SDK beta. Contains references to books and newsgroups.

- Dino Esposito, ***The Component Model in ASP.NET***;
  http://msdn.microsoft.com/msdnmag/issues/01/02/cutting/cutting0102.asp; February 2001
  Explanation of the extensible object model and server controls, including code samples.

- Sun Microsystems; ***Comparing JavaServer Pages and Microsoft Active Server Pages Technologies***; http://java.sun.com/products/jsp/jsp-asp.html; November 1999
  Sun stresses the advantages of JSP over ASP, such as platform independence and cross-platform reusability, and web server independence. Sun also mentions the use of Java for scripting, instead of Jscript and VBScript, which offers a richer library and better maintainability.

## 6.2 Java versus C#

The goal of this section is not to provide a detailed comparison of Sun's Java and Microsoft's C#. It rather summarizes the key points of many articles that have been published on this subject.

At this point in time (March 2001), the C# SDK is only available as a beta version, and documentation is only sparsely available. Although C# shows some promising features that go beyond Java's current language specification, its stability, usability and security have yet to be proven in a larger scale.

The developer community is reserved in estimating C#'s impact on Java. A survey conducted by IBM shows (September 2000) that most developers don't believe in a significant impact of C# on Java programming. However, many developers believe that the introduction of C# will stimulate Sun to more actively improve Java. Furthermore, C# is expected to become the primary development language for web application development under the Windows platform.

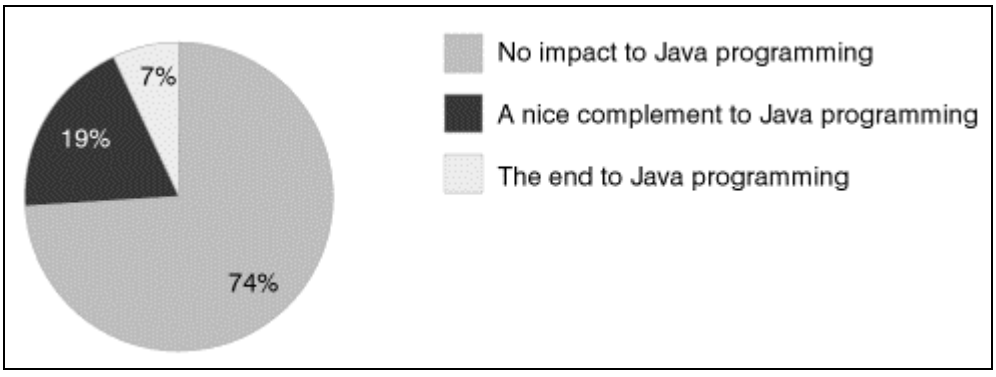


**Figure 19: Survey Results: Impact of C# on Java Programming**

Source : http://www-106.ibm.com/developerworks/library/java-pollresults/csharp.html

### 6.2.1 The C# Language Specification – an Overview

C# was introduced by Microsoft in 2000 as “the premier language for writing .Net applications in the enterprise computing space”, as Microsoft puts it. C# derives from C and C++. However, it modernizes C++ in the area of classes, namespaces, method overloading, and exception handling. Certain features, such as macros, templates, and multiple inheritance have been eliminated from the language, to make its use easier and less error-prone. C# is clearly Microsoft’s answer to Java (Sun prevented Microsoft from integrating Microsoft’s Java implementation “Visual J++” more closely to the Windows environment through a lawsuit). The resemblance to Java is very strong; however, Microsoft tries to offer an even more powerful language to the web application programmer under the Windows platform. Java and C# are compared in detail in Chapter 6.2.3.

### 6.2.2 The .NET Common Language Runtime (CLR)

Microsoft recognized the advantages of Sun Microsystems’ approach of generating so-called “byte code”, the platform-independent semi-compiled code that is generated by the Java Compiler (javac) and executed through the Java Virtual Machine (JVM). Microsoft’s version of this technology is implemented as the Common Language Runtime (CLR) within the .NET platform. Similar to the Java platform, this runtime manages the execution of code via the Virtual Execution System (VES), and it provides services that make programming easier and faster (Rapid Application Development concepts). The managed code that is executed by the CLR is called “Intermediate Language” (IL). C# is compiled to IL, but other programming languages, such as Visual Basic or C++ can be compiled to IL as well. Programming is made easier through automatic memory management (garbage collection), cross-language integration (including exception handling), and versioning support.

### 6.2.3 Comparison of Java and C#

Both C# and Java are languages that resemble C++ and C, but have cleaned up with some of the less used (such as templates) or error-prone (such as multiple inheritance) features of C++. Following is a table that compares the two language’s features in detail.

| Feature | Java | C# |
|---|---|---|
| Lack of pointers | Yes | Yes |
| Single operator, the “dot” | Yes | Yes |
| Unified type system<br>(e.g. no varying range of the integer type) | Yes | Yes |
| Boxing/un-boxing of variables | No | Yes |
| Enumerations | No | Yes |
| Garbage collector | Yes | Yes |
| Exception handling (try, catch, finally) | Yes | Yes |
| Security model | Yes | Yes |
| Data encapsulation (class model) | Yes | Yes |
| Inheritance (from a single class only) | Yes | Yes |
| Polymorphism | Yes | Yes |
| Fully object oriented<br>(all functions and variables are contained in classes) | Yes | Yes |
| Implicit method overriding prohibited | No | Yes |
| Implementation of Interfaces | Yes | Yes |
| Emulation of function pointers<br>(with interfaces or “delegates”) | Yes | Yes |
| Operator overloading | No | Yes |
| Enforced type safety: | | |
| Checking of casts for validity | Yes | Yes |
| Use of un-initialized variables not permitted | Yes | Yes |
| Array bounds checking | Yes | Yes |
| Overflow checking for arithmetic operations | Yes | Yes |
| Support of versioning | No | Yes |
| Pre-processor (conditional compilation) | No | Yes |
| Automatic creation of documentation files | Yes | Yes |
| Interoperation with other languages (C++, VB) | Yes | Yes |
| Interoperation with native code | Yes | Yes |

**Table 8: Comparison of Java and C#**

One can see from Table 8 that Java and C# have many things in common. C#'s syntax sometimes allows a slightly more elegant way of coding (see references). At this point in time, the C# language specification has a few additional features compared to Java. However, Java offers a large amount of class libraries and development support that C# cannot offer yet.

### 6.2.4 Comparison of the Java Runtime and the CLR

The core part of the Java Runtime Environment and the Common Language Runtime are the execution systems, which are the Java Virtual Machine (JVM), and the Virtual Execution System (VES), respectively. Following is a table that compares the features of the two.

| Feature | JVM | VES |
|---|---|---|
| Interpretation of intermediate code from various languages | No* | Yes |
| Class loader enforcing security<br>(Consistency and accessibility checks) | Yes | Yes |
| Garbage collection, exception handling, stack tracing | Yes | Yes |
| Management of threads and contexts | Yes | Yes |
| Many platforms supported | Yes | No |

**Table 9: Comparison of JVM and VES**

* Third party vendors offer language translators to Java.

The VES will be available initially only on Windows operating systems, and other platforms are unlikely to be supported in the near future. Java Virtual Machines, on the other hand, are available on many different platforms, such as Windows, Macintosh, Unix, or Linux.

### 6.2.5 Selected References

- Christoph Wille; ***Presenting C#***; July 2000
  This book introduces the language C# and its features. Comparisons are often drawn to C and C++. The book also describes the Microsoft .NET elements from a developer's point of view, including the Common Language Runtime, the Intermediate Language, and the VES. Unfortunately, security issues, which are a main concern in developing web applications, are not addressed.

- Mark Johnson; ***C#: A Language Alternative Or Just J--?***;
  http://www.javaworld.com/javaworld/jw-11-2000/jw-1122-csharp1.html; November 2000
  A detailed comparison of Java and C# with many programming examples. Java and C# code is directly compared in tables. Includes a long list of additional resources.

- Jim Farley; ***.NET vs. J2EE: How Do They Stack Up?***;
  http://java.oreilly.com/news/farley_0800.html; November 2000
  Compares Microsoft's .NET and the Java 2 Enterprise Edition. The main technologies that are part of the two frameworks are listed, and key differentiators are presented. Possible impacts of .NET on the Windows and the Java developer community are discussed.

- Michael L. Perry; ***C#, The Natural Progression***;
  http://www.javaworld.com/javaworld/jw-08-2000/jw-0804-itw-csharp.html; July 2000
  Information on the chief developer of C#, Anders Hejlsberg. Overview of C# features, especially regarding RAD. Reasons why Microsoft created another programming language.

- Anders Hejlsberg, Scott Wiltamuth; ***C# Language Specification***;
  http://msdn.microsoft.com/library/default.asp ; 2000

# 7 Conclusions

With the introduction to the idea of Open Source and the overview of available products, the author has introduced the reader to the Open Source world, which is currently very much alive. Developers can chose from a growing variety of Open Source products to assemble a development environment that fits their needs. The utilization of Java as the primary development language for Open Source Web technologies results in platform-independence and high reusability of the developed components. Furthermore, Open Source products interoperate well (e.g. Apache and Tomcat) and they are often more configurable than their commercial counterparts.

This thesis has demonstrated that Open Source Web application development can hold up to the high expectations. The case study presented in Chapter 4 has confirmed that Linux, CVS, and Tomcat, and MySQL can provide the means to develop and deploy a Web application in a free-of-charge environment while providing high reliability, usability, and security.

Development using Open Source products is without a doubt more difficult than with its commercial counterparts. This is mainly because documentation is not as readily available. This may change in the future, though, due to the growing popularity of Open Source.

To widen the reader's perspective, a comparison of the Web enabling Java technologies by Sun Microsystems and their most important current and prospective competitor, Microsoft's ASP and .NET, has been presented. Both approaches have many similarities, even more so with the introduction of the .NET platform. However, since ASP and C# cannot offer significant advantages at this point in time and lack platform independence, it seems likely that Java will continue to be the dominating programming language for Web applications in the near future.

The author hopes to having provided a good starting point for newcomers to Web application development using Open Source and Java with this document. Any comments or questions are very welcome; please email klimke@alum.mit.edu.

# 8 Appendix

## 8.1 Tips on How to Learn Linux Quickly

This section's purpose is not to give an introduction to Linux, but rather to advice a Linux newcomer on how to learn to use and understand the environment quickly.

- Take a look at the various tutorials and quick references available on the Internet, e.g. this one by Mark Allen: http://ctdp.tripod.com/os/linux/usersguide/index.html
- On many topics, there are so-called "How-To" documents available. A complete list of all How-Tos can be found at http://www.linuxdoc.org/HOWTO. These documents are quite comprehensive and usually maintained well.
- Don't use the graphical desktop environments such as Gnome or KDE. They hide a lot of details from the user and don't foster the understanding of the Linux operating system.

The following commands help to quickly get information on Linux shell commands:

- The *whatis* command can be used to get a short description of any system command.
  Example: *whatis mkdir* will display a brief description of the *mkdir* command.
- Linux system commands or files usually come with a manual file. These manuals are very detailed. They include the complete syntax of commands, or the file format. The manuals can be accessed with the *man* command from the Linux shell.
  Example: *man cp* displays the manual for the Linux file copy command *cp*.
- When not yet familiar with command names, the right command can often be found quickly with *man –k {keyword}*. This command browses the internal Linux command descriptions database for the given keyword, and displays all matches.
  Example: *man –k delete* will display all commands that contain the word "delete" in their *whatis* description.
- System commands can be located with the *whereis* command (only system commands; for other files, use *slocate*). If the search is successful, a list of all location paths is returned.
  Example: *whereis linuxconf*
- Other files can be located with the *slocate* command. The search returns all path or file names containing the search keyword.
  Example: *slocate tomcat.sh*

## 8.2 Glossary

ASP Active Server Pages – Microsoft's technology supporting dynamic web content. ASP allows server-side programming in VBScript or JavaScript.

BLOB A BLOB is a database data type for storing binary large objects.

CGI The Common Gateway Interface common gateway interface is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user. CGI is the basis for most dynamic content generation languages today.

COM Microsoft's Component Object Model serves as a framework for the interaction of distributed objects in a network, similar to CORBA.

CORBA The Common Object Request Broker Architecture is a framework for the creation, distribution, and management of objects in a network.

CSS Cascading Style Sheets are a technology to define and apply formatting to HTML pages.

DOM The Document Object Model is a programming interface specification developed by the W3C. The DOM serves to represent XML or HTML documents as an object with a hierarchical structure, so it can be programmatically modified.

IDE An Integrated Development Environment allows editing, compilation, and debugging within a single, comprehensive user interface.

NCSA The National Center for Supercomputing Activities, located at the University of Illinois at Urbana-Champaign

PHP PHP is widely used server-side, cross-platform, HTML embedded scripting language. The abbreviation PHP originated from the first versions of the PHP parser in 1995 known as the Personal Home Page Tools.

RPM RPM stands for the "Red Hat Packaging" format. RPM allows for the packaging of software (source code and binaries) into archives, so that the binaries can be easily installed and the source code can be reconstructed.

Tag A tag is a generic term for a language element descriptor. For example, tags are used in HTML to define different text elements (e.g. the <h1> indicates a heading level 1, <table> indicates a table). While HTML has a fixed set of tags with specific meaning, other languages such as XML allow for the creation of new tags with customized behavior.

W3C The World Wide Web Consortium is a neutral organization, which seeks to promote standards to further evolve the Web.