INVESTIGATING THE J2EE SOFTWARE ARCHITECTURE FOR INFRASTRUCTURE MONITORING: A WATER METERING CASE STUDY

By
MAMEET KHANOLKAR

Bachelor of Engineering, Chemical Engineering
University Of Mumbai, India (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, Civil and Environmental Engineering
Thesis Supervisor

Accepted by: _____
Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies

INVESTIGATING THE J2EE SOFTWARE ARCHITECTURE FOR INFRASTRUCTURE MONITORING: A WATER METERING CASE STUDY

By
MAMEET KHANOLKAR

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

MASTER OF ENGINEERING
In Civil and Environmental Engineering

Abstract

Java 2 Enterprise Edition (J2EE) isn't a compiler or programming language. It is a platform for mainframe-scale computing typical of a large enterprise. Infrastructure applications like billing systems and monitoring systems are examples of enterprise systems. J2EE was developed by Sun Microsystems and its associates as a Java-centric platform for enterprise computing.

Distributed applications require access to a set of enterprise services. Typical services include transaction processing, database access and messaging. The J2EE architecture unifies access to such services in its enterprise service (Application programming interfaces) API's. In J2EE the applications programs can access these API's via the web container.

The Department of Public-Works of the town of Arlington, Massachusetts is installing a new wireless meter reading system to read all of its water meters multiple times per day. With the change in the system and also with the very large amount of data coming in from the water meters daily, a water meter management was built which could perform analysis on the data that is received through the wireless system and could guide operational and planning decisions.

Thesis Supervisor: George Kocur
Title: Senior Lecturer in Civil & Environmental Engineering

Acknowledgement

The following are the people who I wish to thank for this thesis:

Prof: George Kocur, who was my graduate advisor and also my thesis advisor, for being very supportive and understanding in what I was trying to achieve via this thesis.

Mr. Kent Larson of the Department of Public-Works of Arlington, Massachusetts for all his help in the Master of Engineering (M.Eng) project, which is used as a case study in this thesis.

Bradford Butler and Sebastian Bogershausen who were my project partners for the M.Eng project at Arlington. Sincerest thanks to them for being very professional team members and friends.

The M.Eng IT students (year 2000-2001) who have been a constant source of motivation throughout this entire experience at MIT

My fiancé Ms Rachna Jotwani for providing love and emotional support in my life.

My sister and brother-in-law, who live here in Boston, for additional family support during my time at MIT.

Lastly, I am forever grateful to my parents back home in India for their support in my life and for bringing me to where I am today.

Table of Contents

Table of Figures

Introduction

This thesis discusses the Java 2, Enterprise Edition (J2EE) platform architecture and investigates into how it can be applied to infrastructure monitoring applications. Infrastructure applications could mean any kind of software application that is used to control or implement infrastructure related activities like management, monitoring and billing for electricity, water. These applications could be built for a small town or may be for an entire city or country depending on the application.

In chapters 1 and 2 the thesis gives an overview of the J2EE platform and the technologies involved in the specifications. These chapters give the definitions and some basics of all the technologies involved and then provide some insight into the overall framework required for deploying applications built on the J2EE platform.

For the case study the thesis will discuss the Arlington Water Project, which is an ongoing Master of Engineering Project (2001) in the information systems group of the Civil & Environmental Dept at MIT. The project is a collaborative effort between MIT and the 'Department of Public-Works' (DPW) of the town of Arlington. The project involved building a customized software application to help the DPW to manage its water supply system and also do analysis on the data that is coming in very frequently from the new wireless meter reading system that would be installed in the new future.

The thesis introduces the project briefly in chapter 3. Chapter 4 discusses the basic J2EE technologies used in the project and also goes into the basics of using those technologies in application development. Chapter 5 uses one of the functionalities namely the Bill Preparation – Abnormal Increase (BPAI) to explain how the different technologies under J2EE work together to implement it. This chapter explains the functionality in 2 perspectives, the user perspective and the code perspective. The user perspective is what the user sees when he runs the functionality on the system. The code perspective is what happens at the back end in the system and how the code is structured so that the system actually does the analysis corresponding to the functionality. As a reader of the thesis it is important to note that all the other functionalities in the project are implemented in exactly the same way or framework that was used for the implementation of the BPAI functionality.

In chapter 6 the thesis looks at an alternative to using J2EE in Enterprise application development ie. the .NET platform provided by Microsoft. This chapter is meant to give the reader an overall perspective of the two leading platforms that are currently in the competition for building enterprise-wide software applications. This chapter differentiates between the two platforms and presents a conclusion on when it might be preferable to use either of the two options.

1 JAVA 2 Platform, Enterprise Edition (J2EE) An Introduction

1.1 Genesis

J2EE is a platform developed by Sun Microsystems and its associates for making Java a platform for enterprise computing. J2EE is a platform for mainframe-scale computing typical of a large enterprise.

In this chapter, the thesis gives a brief introduction of the J2EE architecture and the various Java technologies that are part of its specification.

1.2 Need for J2EE in Enterprise Application Development

With the advent of the Internet and a gradual shift to an information economy, many businesses are rethinking their basic business practices. One place these shifts in business practices have been felt is at the application development level. The whole revolution is driven by the rapidly changing technological and economic landscape, which has created some new challenges for today's enterprise application developer.

Some of the application development challenges are listed below:

- **Responsiveness**
  Responding quickly to new directions and information is critical in establishing and maintaining a competitive edge.

- **Programming productivity**
  Direct adoption of new technologies is insufficient unless they are properly utilized to their full potential and appropriately integrated with other relevant technologies. Thus, the ability to develop and then deploy applications as effectively and as quickly as possible is also important.

- **Reliability & Availability**
  In the Internet economy downtime can be fatal to the success of a business. The ability to get web-based operations up and running, and to keep them running is critical to success. The application must also be able to guarantee the reliability

of business transactions so that they will be processed completely and accurately.

- **Security**

  The Internet has not only exponentially increased the number of potential users but also the value of a company's information, thus the security of that information has become a prime concern. As technologies become more advanced, applications more sophisticated, and enterprises more complex, the ability to implement an effective security model becomes increasingly difficult.

- **Scalability**

  The ability for the application to grow to meet new demands both in its operation and user base is important when an applications potential user base may be millions of individual users through the Internet. To scale effectively requires not only the ability to handle a large increase in the number of clients but also effective use of system resources.

- **Integration**

  Although information has grown to be a key business asset, much of this information exists as data in old and outdated information systems. In order to maximize the usefulness of this information, applications need to be able integrate with the existing information systems. The ability to combine old and new technologies is key to the success of developing for today's enterprises.

1.3 System Architecture

The thesis will now introduce the concept of *n-tier architecture* before discussing the J2EE platform.

1.3.1 2-Tier Architecture

In a 2-Tiered architecture there is a clear separation between the data and the presentation/business logic. The application exists entirely on the client machine while the database server is deployed somewhere in the organization.

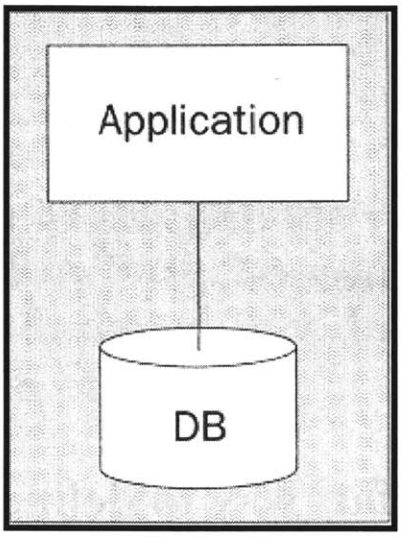

Figure 1: 2-tier Architecture

The processing load is given to the client PC while the database (DB) server simply acts as a traffic controller between the application and the data. The application performance tends to suffer due to the limited resources of the PC. When the entire application is processed on a PC, the application is forced to make multiple requests for data before even presenting anything to the user. These multiple database requests can heavily tax the network.

Another problem of the 2-Tiered approach is with maintenance. Even the smallest of changes to an application might involve a complete rollout of the entire user base. Even if it is possible to automate the process, you still have to update every client installation.

1.3.2 3-Tier Architecture

Ina 3-Tier architecture the application is broken up into 3 separate logical layers, each with a well-defined set of interfaces.

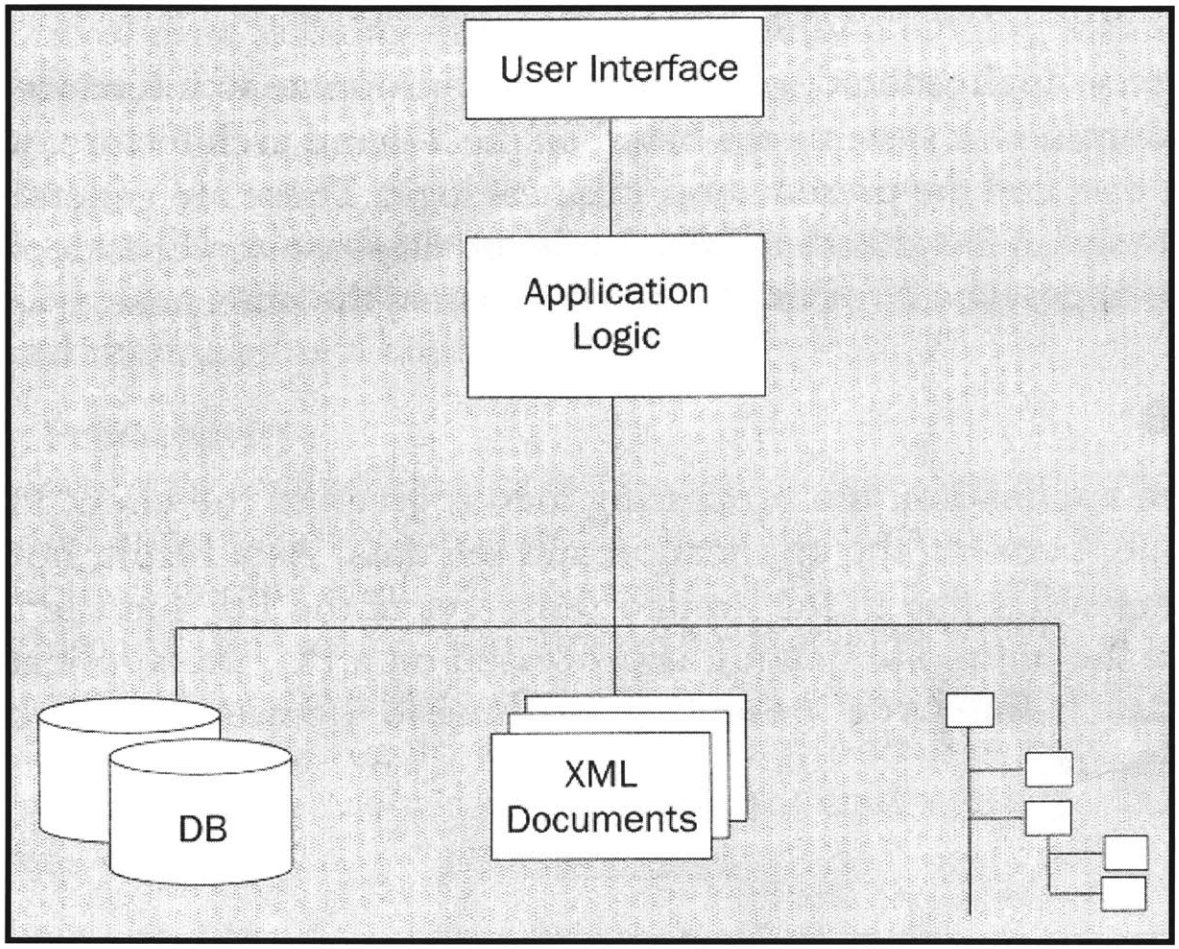

Figure 2: 3-tier Architecture

The first tier is referred to as the 'Presentation layer' and typically consists of a graphical user interface of some kind. (eg. Browser).

The middle tier or business layer consists of the application or business logic.

The third tier – the data layer – contains the data that is needed for the application.

The middle tier is basically the code that the user calls upon through the presentation layer to retrieve the desired data. The presentation layer receives data and formats it for display on the presentation layer. This separation of application logic from the user interface adds enormous flexibility to the design of the application. Multiple user-interfaces can be built and deployed without ever changing the application logic, provided the application logic presents a clearly defined interface to the presentation layer.

The third tier data can consist of any source of information, including an enterprise database such as Oracle or Sybase, a set of XML *(Ref: sec 1.10.9)* documents or even a directory service.

1.3.3 n-Tier Architecture

As the title suggests, there is no hard and fast way to define the application layers for an n-tier system. In fact, an n-tier system can support a number of different configurations. In an n-tier application, the application logic is divided by function rather than physically.

The n-tier architecture can be broken down into

- A user interface (UI) that handles the users interaction with the application. This can be a web browser running through a firewall*, a heavier desktop application or even a wireless device.

  *A firewall is a set of related programs, located at a network gateway server that protects the resources of a private network from users from other networks.
- Presentation logic that defines what the user interface displays and how a users requests are handled. Depending on the different user interfaces that the application supports you may have different versions of the presentation logic to handle the client properly.
- Business Logic that models the applications business rules, often through the interaction with the application's data.
- Infrastructure Services that provide additional functionality required by the application components, such as messaging, transactional support etc
- The data layer where the enterprise data resides.

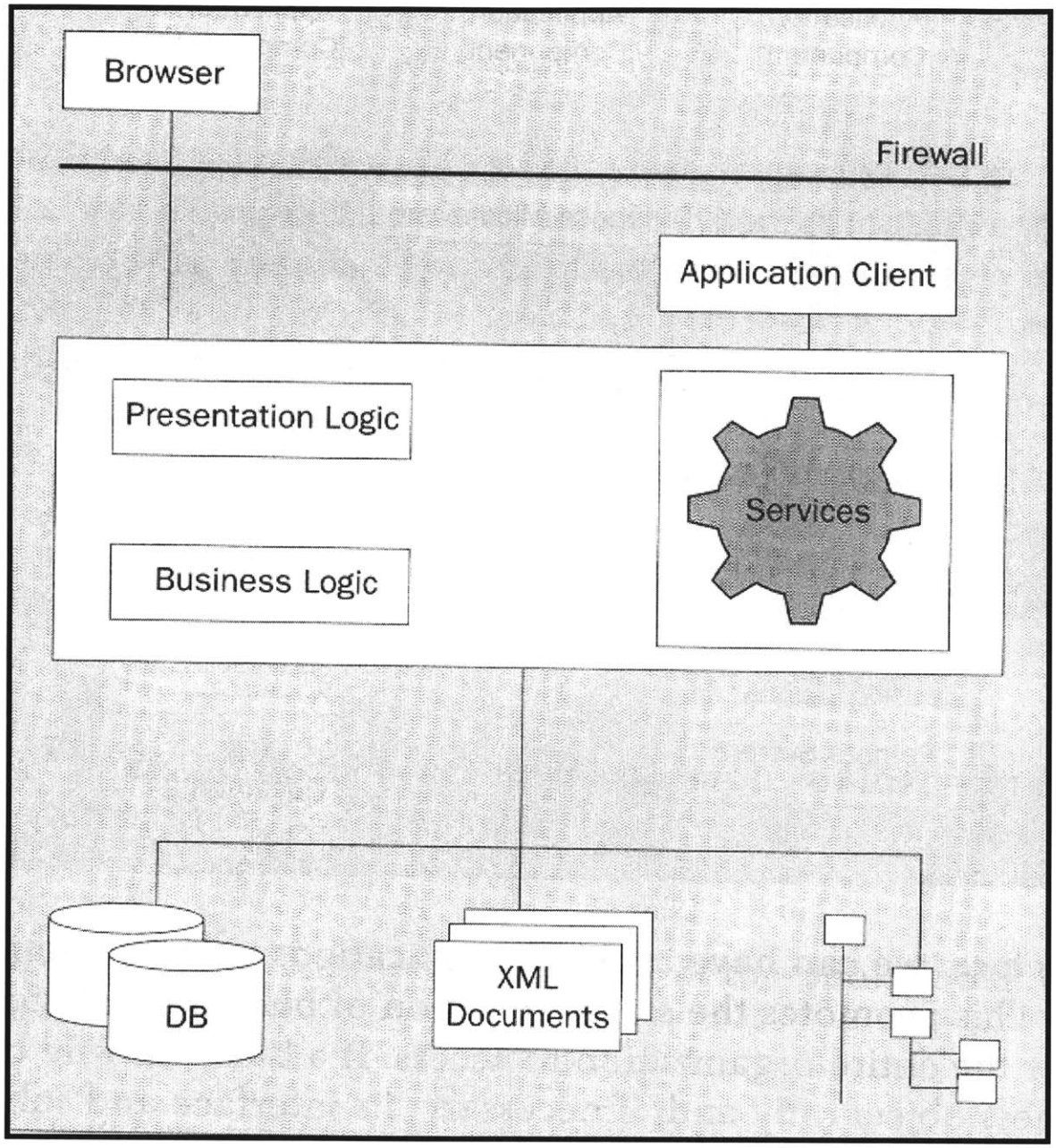

Figure 3: n-tier Architecture

1.3.4 Enterprise Architecture

In an Enterprise, there are many different applications –possibly with different architectures and they all need to be able to communicate with each other.

The n-tier architecture described earlier was application specific. Rather than a change in architecture – enterprise architecture is basically just n-tier – but with a change in perception. To turn the n-tier system into an enterprise system, the middle tier is

extended to allow for multiple application objects rather than just a single application. These applications each must have an interface that allows it to work together with the others.

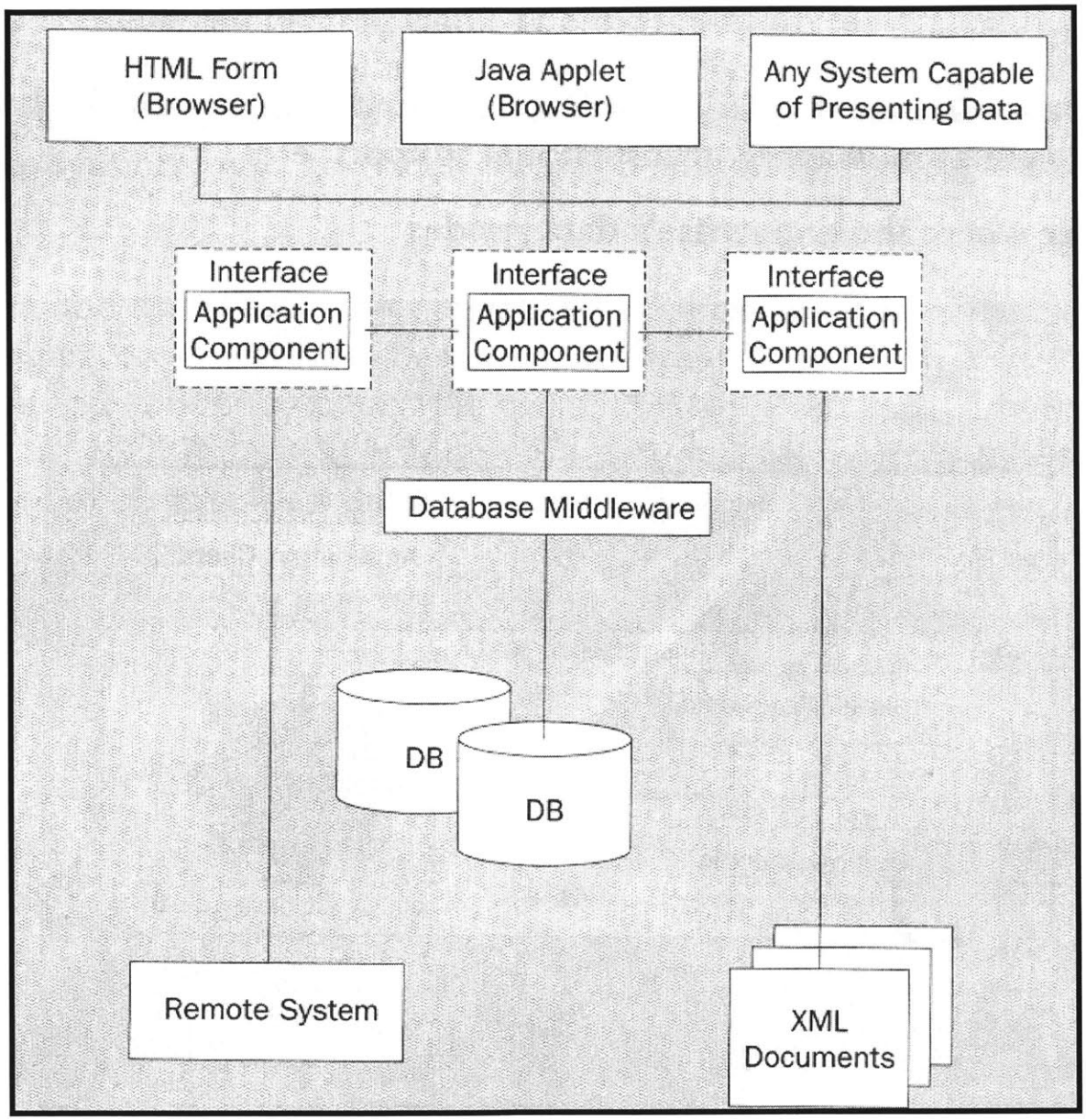

Figure 4: Enterprise Architecture

An interface can be thought of as a contract. Each object states through its interface that it will accept certain parameters and return a specific set of results. Application objects communicate with each other using their interfaces.

In enterprise architecture, multiple applications use a common set of components across an organization.

1.4 Java language in development of Enterprise applications

Platform Independence

An enterprise's information is spread disparately across many platforms and applications. It is important therefore, to leverage a programming language that can work equally well throughout the enterprise without having to resort to awkward, inefficient translation mechanisms. Java's platform independence capability allows it to be distributed throughout an enterprise on different kinds of platforms.

Reusability

Code reuse is essential to all programming. Segregating an application's business requirements into component parts is one way to achieve reuse; using object-oriented concepts to encapsulate shared functionality is another. Java uses both. Java is an object oriented programming language and as such, provides mechanisms for reuse.

Modularity

When developing a complete server side application, programs can get large and complex very quickly. It is always best to break down an application into discreet modules that are each responsible for a specific task. 'Java Servlets', 'Java Server Pages' and 'Enterprise Java Beans' provide ways to modularize an application - breaking your application into tiers and tasks.

The goal behind the J2EE platform is to provide a simple, unified standard for distributed applications through a computer based application model.

1.5 J2EE Platform

The J2EE platform is essentially a distributed application server development – a Java environment that provides the following –

- A runtime infrastructure for hosting applications
- A set of Java API's to build applications

1.5.1 J2EE Runtime

The J2EE bundles together API's that have been in existence in one form or another for quite some time. The most significant aspect of J2EE is its abstraction of the runtime infrastructure. The J2EE specification does not specify how a J2EE runtime should/could be built. Instead, the J2EE specifies roles and interfaces for applications, and the runtime onto which applications could be deployed. This results in a clear demarcation between applications and the runtime infrastructure. This demarcation allows the runtime to abstract most of the infrastructure services that enterprise developers traditionally build up on their own.

As a result, using J2EE, a developer focuses on the application logic and related services, while leveraging the runtime for all infrastructure related services.

Apart from specifying a set of standard API's, the J2EE architecture provides a uniform means of accessing these services via its runtime environment.

As mentioned earlier J2EE does not specify the nature and structure of the runtime. Instead, it introduces what is called a container, and via the J2EE API's specifies a contract between containers and applications.

1.5.2 J2EE API's

Distributed applications require access to a set of enterprise services. Typical services include transaction processing, database access, messaging etc. The J2EE architecture unifies access to such services in its enterprise service API's. In J2EE the applications programs can access these API's via the container.

The specification of the J2EE platform defines a set of Java standard extensions that each J2EE platform must support.

- Java Database Connectivity (JDBC) 2.0 Extension
- Remote Method Invocation over the Inter-ORB Protocol (RMI-IIOP) 1.0
- Enterprise Java Beans (EJB) 1.1
- Java Servlets 2.2
- Javaserver Pages (JSP) 1.1

- Java Message Service (JMS) 1.0
- Java Naming and Directory Interface (JNDI) 1.2
- Java Transaction API 1.0
- Java Mail 1.1

All the above API's are specifications, independent of implementation. One should be able to access services provided by these API's in a standard way, irrespective of how they are implemented.

1.6 J2EE Architecture - Containers

A typical commercial J2EE platform includes one or more containers. A J2EE container is a runtime to manage applications components and to provide access to the J2EE API's. Beyond the identity associated with the runtime, J2EE does not specify any identity for the containers.

The following figure shows the architecture of J2EE:

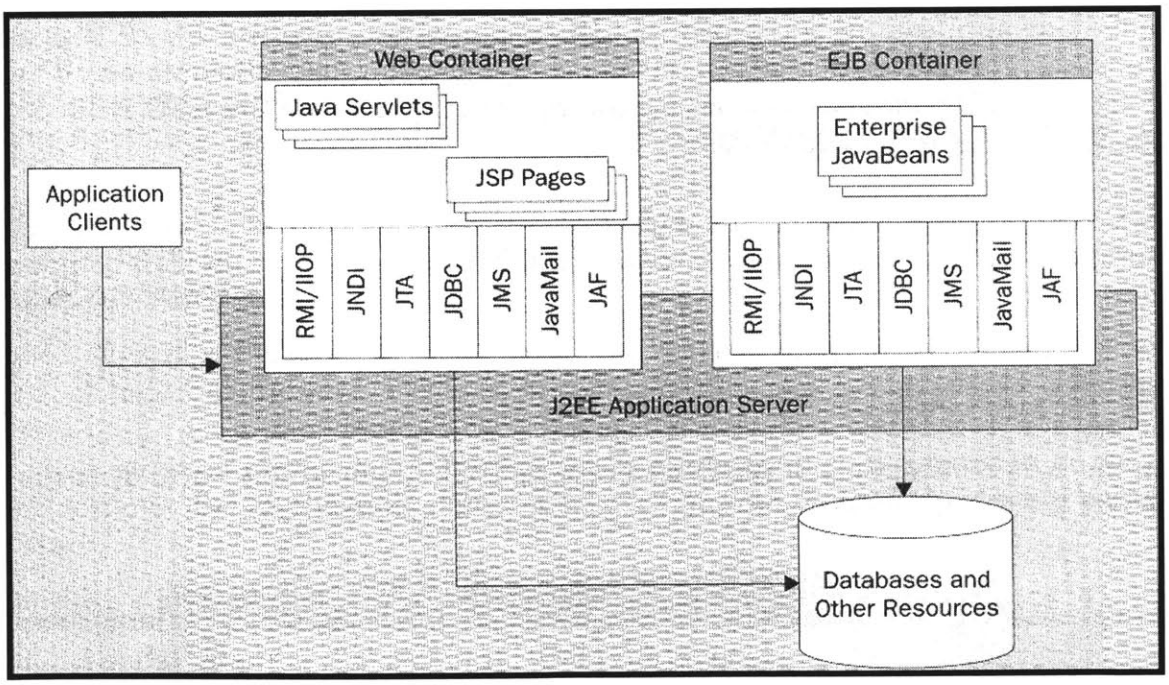

Figure 5: J2EE Architecture

The architecture shows two containers:

1. A web container for hosting Java servlets and JSP pages.
2. An EJB container for hosting enterprise javabean components

Note: Apart from the 2 containers mentioned above, J2EE also specifies 2 more containers – an applet container to run applets, and an application client container for running standard Java application clients. This thesis's focus is limited to web and EJB containers only.

In the figure above, the vertical blocks at the bottom of each container represent the J2EE API's. Apart from access to the infrastructure-level-API's, each container also implements the respective container- specific API (Java servlet API for the web container and the EJB API for the EJB container)

The stacks of rectangles (servlets, JSP pages, and EJB's) in the figure are the programs that a developer writes and hosts in these containers. In the J2EE parlance, these programs are called application components.

So, a container is a Java 1.2(Java 2 Standard Edition 1.2) runtime for application components.

In this architecture, there are primarily 2 types of clients

- **Web clients**

  Web clients normally run in web browsers. The user interface is generated on the server side as HTML or XML, and is downloaded and then rendered by the browsers. These clients use HTTP to communicate with web containers. Application components in web containers include Java servlets & Java server pages. These components implement the functionality required for the web clients. Web containers are responsible for accepting the requests from the clients, and generating responses with the help of the application components.

- **EJB clients**

  EJB clients are applications that access EJB components in EJB containers. There are 2 possible EJB clients. The first category is application clients. Application clients are stand-alone applications accessing the EJB components using the RMI-IIOP protocol (Ref to sec 1.10.5.0). The second category, are the components of the web container. Java servlets and JSP pages can also access

the EJB components via the RMI-IIOP protocol in the same way as the application clients.

In either case, clients access the application components via the respective container. Web clients access the servlets and JSP pages via the web container, and EJB clients access the EJB components via the EJB container.

1.7 Container Architecture

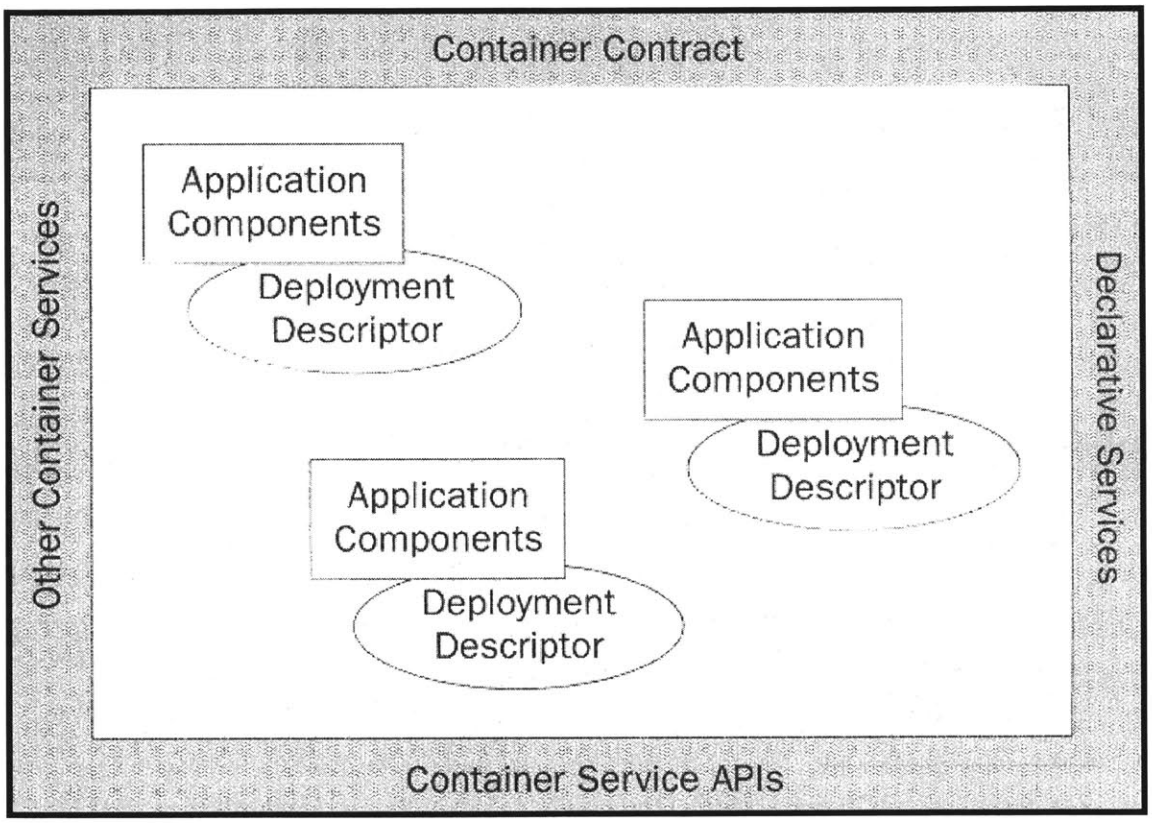

Figure 6: Container Architecture

In this architecture, you provide the following as a developer

1.7.1 Application Components

Application components include the servlets, JSP's, EJB's etc. In J2EE the components can be packaged into archive files.

1.7.2 Deployment Descriptors

A deployment descriptor is an XML file (Ref sec 1.10.9) that describes the application components. It also includes additional information required by containers for effectively managing application components.

The rest of the figure forms the container. The architecture of a container can be divided into 4 parts.

1. **Component Contract:**
   A set of API's, specified by the container that the application components are required to extend or implement.

2. **Container Service APIs**
   Additional services provided by the container, which are commonly required for all applications in the container.

3. **Declarative Services**
   Services that the container interposes on your application, based on the deployment description for each of the application component.

4. **Other Container Services**
   Runtime services related to component lifecycle, resource pooling, garbage collection etc.

1.8 J2EE Technologies

These are the collection of technologies (Java API'S and Associated technologies) that provide the mechanics needed for building large, distributed enterprise applications.
The large collection of quite disparate technologies can be divided according to use:

1.8.1 The Component Technologies

These technologies are used to hold the most important part of the application – the business logic. The J2EE platform provides three technologies for developing components. (The J2EE platform does not specify that an application need make use of all 3 types).

1) Servlets
2) Java Server Pages (JSP)

3) Enterprise Java Beans (EJB)

1.8.2 Web Components

These can be categorized as any component that responds to an HTTP request. A further distinction that can be drawn is on the hosting container for the application components.

1) **Servlets**

   Servlets are server side programs that allow application logic to be embedded in the HTTP request-response process. Servlets provide a means to extend functionality of the web server to enable dynamic content in HTML, XML or other web languages.

2) **Javaserver pages (JSP)**

   Javaserver pages (JSP) provide a way to embed components in a page, and to have them do their work to generate the page that is eventually sent to the client. A Javaserver page can contain HTML, Java code and javabean components. Javaserver pages are infact an extension of the servlet programming model. When a user requests a JSP page, the web server compiles the JSP page into a servlet. The web server then invokes the servlet and returns the resulting content to the web browser.

   Compared to servlets, which are pure Java code, javaserver pages are merely text-based documents until the web server compiles them into the corresponding servlets. This allows a clearer separation of application logic from presentation logic; this allows application developers to concentrate on business matter and web designers to concentrate on presentation.

1.8.3 Enterprise Javabean Components

The EJB architecture is a distributed component model for developing secure, scalable, transactional and multi-user components. They are reusable software units containing business logic. Just as JSP's allow the separation of application and presentation logic, EJB's allow the separation of application logic from system level services, thus allowing

developers to concentrate on the business domain issues and not system programming. These enterprise beans business objects take 2 basic forms

1) Session Beans
2) Entity Beans

Session Beans

Session beans are of 2 types. A **stateful session bean** is a transient object to represent a client's requests in the application, accessing a database etc. When client operations are completed it is destroyed. The session bean exists for the length of the client session. An example of this is an online shopping cart. Alternatively a **stateless session bean** maintains no state between client requests. Generally, this type of session bean is used to implement a specific service that does not require client state, for instance, a simple database update.

Entity Beans

An entity bean on the other hand is a persistent object that models the data held within the data store. It acts as a object wrapper for the data. Compared to session beans, which can be used by any client, entity beans can be accessed concurrently by many clients but must a unique identity through a primary key. In the J2EE container architecture you can elect whether to have the persistent state of the entity bean managed automatically by the container or whether to implement this manually in the bean itself.

1.9 Service Technologies

Some of the J2EE services for the application components are managed by the containers, so that the developers can concentrate on the business logic. The developer can also programmatically invoke these services when needed.

1.9.1 JDBC

The Java Database connectivity (JDBC) API provides the developer with the ability to connect to relational database systems. J2EE adds an extension to the core JDBC API that comes with the Java 2 Standard edition to add features such as connection pooling and distributed transactions.

1.9.2 Java Transaction API

The Java Transaction API (JTA) is a means for working with transaction, especially distributed transactions independent of the transaction manager's implementation (the Java transaction service (JTS)). In the J2EE platform distributed transactions are considered to be container controlled. However, the J2EE transaction model is somewhat limited so it may be necessary for the developer to implement it.

1.9.3 JNDI

The 'Java Naming and Directory Interface' (JNDI) API in the J2EE platform has two usages

1. It provides the means to perform standard operations with a directory service resource such as LDAP, Novell Directory services or Netscape directory services.
2. J2EE utilizes JNDI to look up interfaces used to create among other things, EJB's and JDBC connections.

1.10 Communication Technologies

The final technology grouping is those technologies that provide the means for the various components and services within a J2EE application to communicate with each other. A distributed application would be pretty ineffectual if these technologies didn't provide the connectivity to hold it all together.

1.10.1 Internet protocols

In n-tier applications, the client will often be a web browser. A client's requests and the servers responses are communicated over 3 main protocols.

1.10.2 HTTP

HTTP or Hypertext Transfer Protocol is a generic, stateless, application-level protocol. It works on a request/response basis – a client sends a request to the server in the form of a request method, URI (Uniform Resource Identifier) and protocol version followed by a MIME message containing request modifiers, client information and possible body content over a connection with a server. The server in turn responds with a status line

followed by a MIME like message containing server information, entity meta-information, and possible entity-body content.

1.10.3 TCP/IP

TCP (Transmission Control Protocol) over IP (Internet Protocol) are actually two separate protocols, but are typically combined into a single entity. IP is the protocol that takes care of making sure that data is received at both endpoints in communication over the Internet. When you type the address of a web site into the browser, IP ensures that your requests and the fulfillment of those requests make it to the proper destinations. For efficiency, the data being sent between the client and a web server is broken into several pieces, or packets. All of these packets do not have to take the same route between the client and the web server. TCP is the protocol that keeps track of all the packets and makes sure that they are assembled in the same order that they were dispatched and are error free. Therefore TCP and IP work together to move the data around on the Internet. For this reason you will see these two protocols combined into TCP/IP.

1.10.4 SSL

Secure Socket Layer (SSL) uses cryptography to encrypt the flow of information between the client and the server. This also provides a means for both parties to authenticate each other. Secure HTTP (HTTPS) is usually distinguished from regular unencrypted HTTP by being served on a different port number, 443, by default.

1.10.5 Remote Object Protocols

In applications where the components are often distributed across many tiers and servers, some mechanism for using the components remotely is required – preferably in a way that client isn't aware that the component is not local to itself.

1.10.5.1 RMI and RMI-IIOP

Remote Method Invocation (RMI) is one of the primary mechanisms in distributed object applications. It allows the use of interfaces to define remote objects. Methods are then called on these remote objects as if they were local. The exact wire-level transportation mechanism is implementation specific.

RMI IIOP is an extension of RMI but over IIOP (Inter –ORB Protocol), which allows you to define a remote interface to any remote object that can be implemented in any language that supports OMG mapping and ORB.

1.10.6 JavaIDL

Through the use of JavaIDL, a Java client can invoke method calls on a CORBA objects. These CORBA objects need not be written in Java but merely need to implement an IDL- defined interface. This is done in conjunction with RMI-IIOP.

1.10.7 JMS

In the enterprise environment, the various distributed components may not always be in constant contact with each other. The Java message provides the mechanism for sending data asynchronously between components. It provides the functionality to send and receive messages through the use of message-oriented middleware (MOM).

1.10.8 JavaMail

An alternative asynchronous process to messaging is JavaMail. JavaMail also allows the sending and receiving of messages; however it is oriented towards the user than parts of an application. JavaMail supports the most widely used Internet protocols like IMAP4, POP#, and SMTP, but compared to JMS its is slower and less reliable.

1.10.9 XML

XML (Extensible Markup Language) influences the way we view, process, transport and manage data. The data description mechanisms in XML mean it is a great way to share information because:

- It is open: XML can be used to exchange data with other users and programs in a platform independent manner.
- It is self-describing which makes it an effective choice for business to business and extranet solutions.
- It allows the sharing of data between programs without prior co-ordination.

XML plays a significant role in the construction of J2EE applications

- The J2EE architecture provides the means for a container to provide services at runtime through the declarative mechanism defined in a deployment descriptor. This deployment descriptor is an XML file.
- XML can be used to integrate a J2EE application with legacy systems.
- Application data can be returned in XML instead of HTML for displaying to the client.

2 Developing J2EE Applications

This chapter will discuss how all the Java technologies introduced in chapter 1 can be packaged together to build enterprise applications.

The J2EE specification specifies the following steps in the application and deployment process.

2.1 Application Component Development

During this step, business rules are modeled in the form of application components. This step ideally could involve using UML to model business logic of the application using class diagrams, Use cases, sequence diagrams, Activity diagrams etc. The next step would involve writing the code for the business model in Java and also with the help of the various JAVA API's for JSP, servlets, JDBC etc.

2.2 Composition of Application components into modules

In this step, the application components are packaged into modules. This phase involves providing deployment descriptors for each module.

A module is used to package one or more related application components of the same type. Apart from the application components, each module also includes a deployment descriptor describing the structure of the module. There are 3 types of modules in J2EE

- **Web Modules**

  A web module is a deployable unit consisting of Java servlets, JSP pages, JSP tag libraries, library JAR files, HTML/XML documents, and other public resources such as images, applet class files, etc. A web module is packaged into a web archive file, also called WAR file. A WAR file is similar to a JAR file, except that a WAR file contains a WEB-INF directory with the deployment description contained a 'web.xml' file.

- **EJB Modules**

An EJB module is a deployable unit consisting of EJB's and associated library JAR files, and resources etc. EJB modules are packaged into JAR files and resources etc. EJB modules are packaged into JAR files, with a deployment descriptor (ejb-jar.xml) in the META-INF directory of the JAR files.

- **Java Modules**
  A Java module is a group of Java client classes packaged into JAR files. The deployment descriptor for a Java module is an application–client.xml file.

2.3 Composition of modules into applications

This step integrates multiple modules into J2EE applications. This requires assembling one or more modules into J2EE applications, and supplying it with descriptor files.

The highest level of packaging is in the form of applications. A J2EE application consist of one or more modules composed into an 'Enterprise Archive' (EAR) file. An EAR file is similar to a JAR file, except that it contains an 'application.xml' file located (located in the META-INF directory) describing the application.

The 'application.xml' is a means of specifying which modules make up the application. The advantage of this structure is that it allows reuse of the various components at different levels. Application components can be reused across multiple web modules. Similarly modules can be reused across multiple applications.

2.4 Application Deployment

In the final step the packaged application is actually deployed and installed on the J2EE platform application server.
This process involves 2 steps

- To prepare the application for installing on to a J2EE application server. This involves copying the EAR files onto the application server,

generating additional implementation classes with the help of the container, and finally installing the application onto the server.

- To configure the application with application server specific information

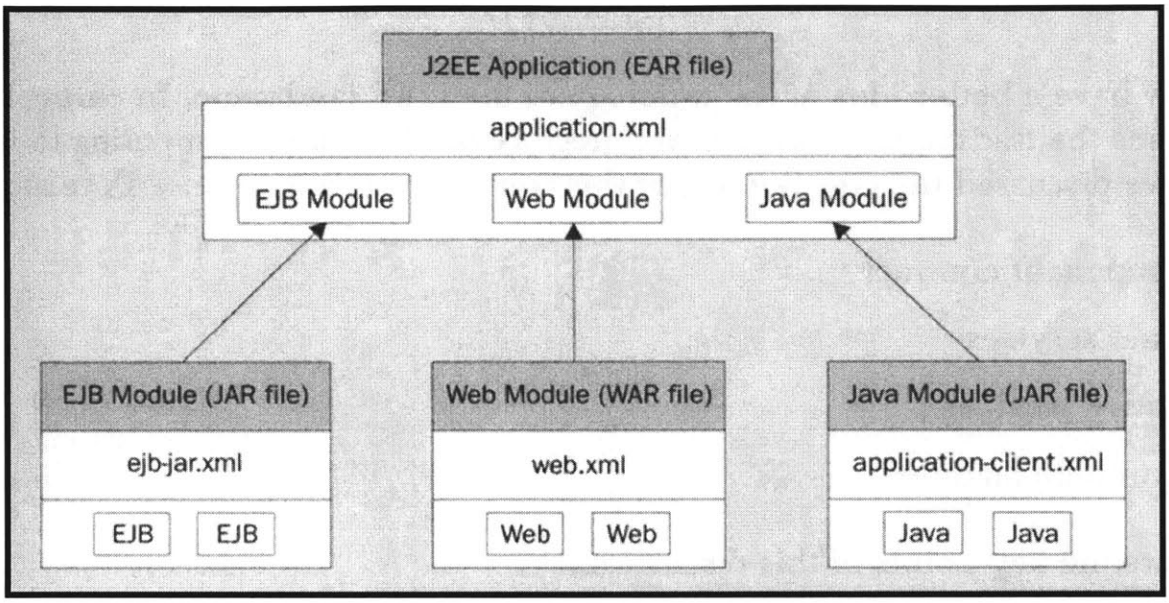

Figure 7: J2EE Application Deployment

2.5 J2EE Application Development and Deployment Roles

Apart from the process, the J2EE specification also defines a number of roles in the development of J2EE applications.

- **J2EE product providers**

  The J2EE product provider provides the base J2EE platform upon which the application is developed – this will be the relevant server vendor who implements the container architecture and the J2EE API's defined by the J2EE specification.

- **Application component Provider.**

  The application component provider is the application developer who creates the application functionality, although this role could be further divided into specific areas of expertise such as web-developer, EJB developer, etc.

- **Application Assembler**

  The application assembler takes the application components and packages them together through the series of modules and descriptor files so that they can be deployed to the production servers.

- **Deployer**

  The deployer installs the packaged application, and configures it for the operating environment on which the application will be running.

- **System Administrator**

  Responsible for maintaining and administering the application once it is deployed

- **Tool Provider**

  Provides tools that are of use in the development and deployment of application components.

3 Water Metering Case Study: Arlington Water Project

3.1 Introduction

The case study for this thesis is a Master of Engineering (M.Eng) project at MIT for the Department of Public Works (DPW) of the town of Arlington, Massachusetts. It involved building a software system for their 'Water Supply Management' division after a new wireless system for reading the water meters would be installed sometime in 2001 or 2001. For further details on the functionalities of the system please refer the project report titled "Water supply management system for the Department of Public Works for the town of Arlington, Massachusetts" (M.eng, Year 2001) *. (See References)

*For further reference please refer to the project report and 2 more thesis documents listed in the references

This thesis will concentrate on the technical challenges in terms of the software system that was built to do analysis on the readings obtained from the wireless system in the town. In particular, one of the functionalities namely 'Bill Prep – Abnormal Increase' (BPAI) would be used to explain the how the J2EE technologies were used to build it.

The project at Arlington will continue for the next year (2001 –2002). Presently the system being built at Arlington is not a distributed system and the thesis will discuss how the J2EE framework could be applied to this project on a distributed computing environment.

The town has an existing system that allows them to bill their customers. The town takes the meter readings (the meter are physically read by personnel) twice a year and consequently the customers are billed twice a year for their water in the town. There are approximately 12,200 water meters in the town.

The new wireless system that will be installed sometime in the near future, will allow the DPW to get readings from the water meters as frequently as every 10 minutes. Even if the town decided to read the meters twice a day, the system would have to deal with approx 24,400 readings on a daily basis as opposed to dealing with approx 24,400 readings every year as per the current system.

The project scope was to build a software system that satisfies the following criteria:

- The system should be able to interface with the wireless system to get the readings when requested
- The system should be able to perform all the functions of the old system
- The system should be able to perform pre-defined analysis (functionalities, see section 3.2) on the data that comes in from the wireless units.
- The system should be user friendly
- The system would be web enabled so that results could be viewed on the web

3.2 Functionalities

With the frequency of readings now on a daily basis, there was pre-defined functionality built into the system that would do some useful analysis on the reading data. For details on each of the functionality please refer to the project report. However, for the purpose of this thesis, a brief description of each is given here.

3.2.1 Bill Preparation

The bill preparation functionality allows the user to review account information for suspect meters before sending billing data to the ICS. The bill prep analysis reports suspect meters with missing readings, estimated readings, abnormal increase in usage, and abnormal decrease in usage. Each of these is separate sub functionality within Bill Prep. Once the user decides all meter information is correct and accurate, he/she can aggregate the daily readings into a monthly value for each meter. The monthly meter information is then sent to the ICS for billing purposes. The following list briefly describes the calculations performed in the bill prep functionality.

3.2.2 Leak Detection

There are two types of analysis used for leak detection.

- The individual leak analysis attempts to identify leaks on the customer side of the meter. The system will analyze individual meter readings to find suspect activity,

which may be the results of a leak. The suspect activity includes sudden increases in use, an average increase in use above a threshold, and constant night use.

- The MWRA comparison analyzes the relationship between the flow of water into the Arlington network, via the MWRA meters, and the consumption of water by the Arlington meters.

3.2.3 Usage Analysis

The usage analysis allows the user to create an aggregate profile for a group of meters. The user will define the group by checking the appropriate parameter check boxes. If a box is checked, the group will include meters with the checked attribute. The aggregate profile will give the user insight on typical historical usage for groups of meters. The usage profile for the group will only span the time period specified by the user.

3.2.4 Water Theft

The water theft analysis attempts to find meters in Arlington with suspicious activity. The analysis looks for four types of suspicious behavior; continuous zero usage, negative usage, broken seals, and loss of physical connection. Meters exhibiting one of the four types of suspicious behavior will be written to the results table when the analysis is run. The user may then take appropriate action for each meter. The available actions include editing information in the database and placing meters on watch.

3.2.5 Meter Watch

The meter watch functionality allows the user to monitor a specific meter. The user must first place a meter on watch using the meter watch configuration page. Next, the user must make sure the meter watch check box is checked before the analysis is run. After the analysis is run, the user may view a report for each meter placed on watch. The report will include a usage profile for each meter.

3.3 System Architecture and J2EE

The entire system for the town was developed at MIT. The system is a web-based system and uses the browser as the UI. The system will reside on the intranet of the DPW and is not accessible to users from outside of the intranet framework.

The technologies used for the various components are as follows

- **Server** – Apache/Tomcat
- **Database**: MySQL (Open Source Database)
- **Programming Language:** JAVA 2, Javabeans (J2EE)
- **Client-side scripting:** Javascript
- **Server side scripting:** JSP & Servlets (J2EE)
- **Web Design:** HTML, FrontPage

As the project was an academic project and also had a limited timeframe, there were some trade-offs made with respect to the system being distributed and also the use of a open source database like MySQL that does not have the functionality that an ORACLE database would have. But, by working around the specific problems the system works efficiently in the current setup.

4 Water Metering Case Study: Basics of the J2EE technologies used in the project

This chapter discusses the basic fundamentals of the various Java technologies used in the case study project. Please note that the explanation given here is only sufficient for the reader to understand the code and design framework used in the case study project only. For a more comprehensive understanding of the technologies, please refer to a book on each of the technologies.

4.1 JSP Basics

Before going into the details of the code and the framework, an understanding of the basics of Java Server Pages (JSP) is necessary.

The goal of JSP is to simplify the creation and management of dynamic web pages by separating content and presentation. JSP's are basically files that combine standard HTML(or XML) and new scripting tags. The objective is that the HTML (or XML) should relate to the presentation (look and feel) of the content on a web page. The content itself could be generated dynamically using the scripting elements.

A JSP page therefore looks somewhat similar to HTML, but it gets translated into a servlet (Refer to sec 1.8.2) the first time it is invoked by a client. Servlets are programs that run on a web server acting as a middle layer between a request coming from the web browser or other HTTP clients and databases or applications on the web server. The resulting servlet from the JSP is a combination of the HTML from the JSP file and embedded dynamic content specified by the new tags.

4.1.1 Writing JSP's

The structure of a JSP page is cross between a servlet and an HTML page, with java code enclosed between the constructs <% and %> and other XML like tags interspersed.

JSP tags fall into 3 categories

4.1.1.1 Directives:

These affect the overall structure of the servlet that results from translation. JSP directives serve as messages sent to the JSP container from the JSP. They are used to set global values such as class declarations, methods to be implemented, output content type etc, and do not produce any out to the client. Directives have scope for the entire JSP file; in other words, a directive affects the whole JSP file but only that file.

Directives start with <%@ and end with %>; the general syntax is:

<%@ directivename attribute ="value" atrribute = "value" %>

For example, if you wanted import the rmi package for the purpose of the JSP page you are writing you would write a directive statement as below

<%@ page import= "java.rmi.*" %>

4.1.1.2 Scripting elements

These let you insert Java code into the JSP page (and hence into the resulting servlet). Java code could be variable or method declarations, scriptlets and expressions.

- **Declarations**

  A declaration is a block of java code in a JSP that is used to define class wide variables and methods in the generated servlet. Declarations are initialized when the JSP is initialized and have 'class' scope in the generated servlet, so that anything defined in a declaration available throughout the JSP to other declarations, expressions and code.

  A declaration block is enclosed between <%! and %> and does not write anything to the output stream

  The syntax is:

  <% Java variable and method declaration(s) %>

  Example:

```
<%!
    int  num = 3;

    public String welcomeMessage(String name)
    {
            return "Hello " + name + "!!";
        }

%>
```

This declares an integer variable called num and a welcomeMessage() method that greets the requested person.

- **Scriptlets**

  A scriptlet is a block of java code that is executed during the request processing time and is enclosed between <% and %> tags. What the scriptlet does depends on the code itself and can be used for producing output to the client. Multiple scriptlets on a JSP are combined in the generated servlet class in the order they appear in the JSP.

  The syntax for the scriptlet is:

  <% Valid Java Code %>

  Example: scriptlet.jsp

```
<html>
 <head>
   <title>Scriptlet Example</title>
 </head>
 <body>
  <h1>Scriptlet Example</h1>

  <%
     for(i=0; i<10 ; I++)
     out.println("<b>Hello World" + i + "</b><br>");

     System.out.println("This is output to the client " + i);
   }
  %>

 </body>
</html>
```

In the JSP above a scriptlet executes a loop 10 times and prints out "Hello World" ten times to the output stream.

- **Expressions**

  An expression is shorthand for a scriptlet that sends the value of a java expression back to the client. The expression is evaluated at HTTP request processing time, and the result is converted to a String and displayed.

  An expression is enclosed in the <%= and the %> tags. If the result of the expression is an object, the conversion is done by using the objects toString() method. The syntax is:

  <%= Java expression to be evaluated %>

  Example: expression.jsp

```
<html>
  <head>
<title>Expression</title>
  </head>
    <body>
  <h1>Expression Page</h1>

  <% !int p=0 ; %>

  <%
    i++;
  %>

  Hello Reader!
  <%= "The value of p is " + p  %>
   </body>
</html>
```

4.2 Using Javabeans with JSP Basics (Reusability)

Instead of putting all the java code in a JSP file an alternative is to put all the business logic code in javabeans and just call the various methods etc in the JSP page. The design concept behind using javabeans with JSP is explained in Ch 5, Section 5.2. javabean is an effective way to separate content from presentation in a JSP. Also, an important concept of reusability is accomplished using javabeans and saves repetition of code.

To discuss the entire javabeans concept in Java is beyond the scope of this thesis. The basics needed to implement them in JSP are discussed here.

A javabean is simple class that you write in Java. A javabean has 3 simple characteristics:

1. A bean must have a zero-argument (empty) constructor
2. A bean class should have no public instance variables
3. Persistent values should be accessed through method getXxx and setXxx. (where xxx is the persistent value being accessed)

4.2.1 Javabean Use

The jsp: useBean action lets you load a bean to be used in the JSP page. The simplest syntax for specifying that a bean should be used in a JSP is:

<jsp: usebean id= "name" class = "package.Class" />

Example:

Suppose we have a Javabean named DbBean.class and it belongs to the package 'Water' then the JSP page where you want use it would have the following:

<jsp: usebean id= "db" class = "Water.DbBean" />

Once the bean is instantiated, you can access its properties with **jsp: getProperty** which takes a name attribute that should match the id given in **jsp:usebean** and a property attribute that names the property of interest. Alternatively, you cold use a JSP expression that explicitly calls a method on the object.

The syntax for accessing the properties of a bean is:

<jsp: getProperty name= "bean id" property= "property name" />

Example:

In the DbBean class if we have a parameter named "url" which we need to access in the JSP once the bean has been instantiated then we could do the following:

```
<jsp: getProperty name= "db" property= "url" />
```

or Alternatively

```
<%= db.getUrl() %>
```

To modify a bean property, we can use **jsp: setProperty.** The syntax is identical to the usage of the jsp:getProperty action. The syntax for setting a bean property is:

```
<jsp: setProperty name= "bean id" property= "property name" value = "value" />
```

Example:

If we need to modify the 'url' property of the DbBean and set it to http://web.mit.edu we could do the following

```
<jsp: setProperty name= "db" property= "url" value= "http://web.mit.edu" />
```

or Alternatively

```
<%= db.setUrl("http://web.mit.edu") % >
```

If you need to run a method named 'connect()' in the DbBean class in your JSP you could execute it within a scriptlet block

Example:

```
<% db.connect()
        ..... (other java code)
%>
```

4.3 JDBC Basics

JDBC is essentially an API for executing SQL statements, and extracting the results. Using this API we can write JSPs that connect to a relational database, execute SQL queries, and process the results extracted.

4.3.1 Database Drivers

A database vendor typically provides a set of API's for accessing data managed by the database server. The 'Java Virtual Machine' uses the JDBC driver to translate the generalized JDBC calls into vendor-specific database calls that the database understands.

The database driver is a piece of software that knows how to talk to the actual database server.

There are several approaches for connecting from an application to database server via a database driver. We will consider the one that was used for the case study project.

Pure Java Driver

These drivers convert the JDBC API calls to direct network calls using vendor-specific networking protocols by making direct socket connections with the database. The figure below shows the process

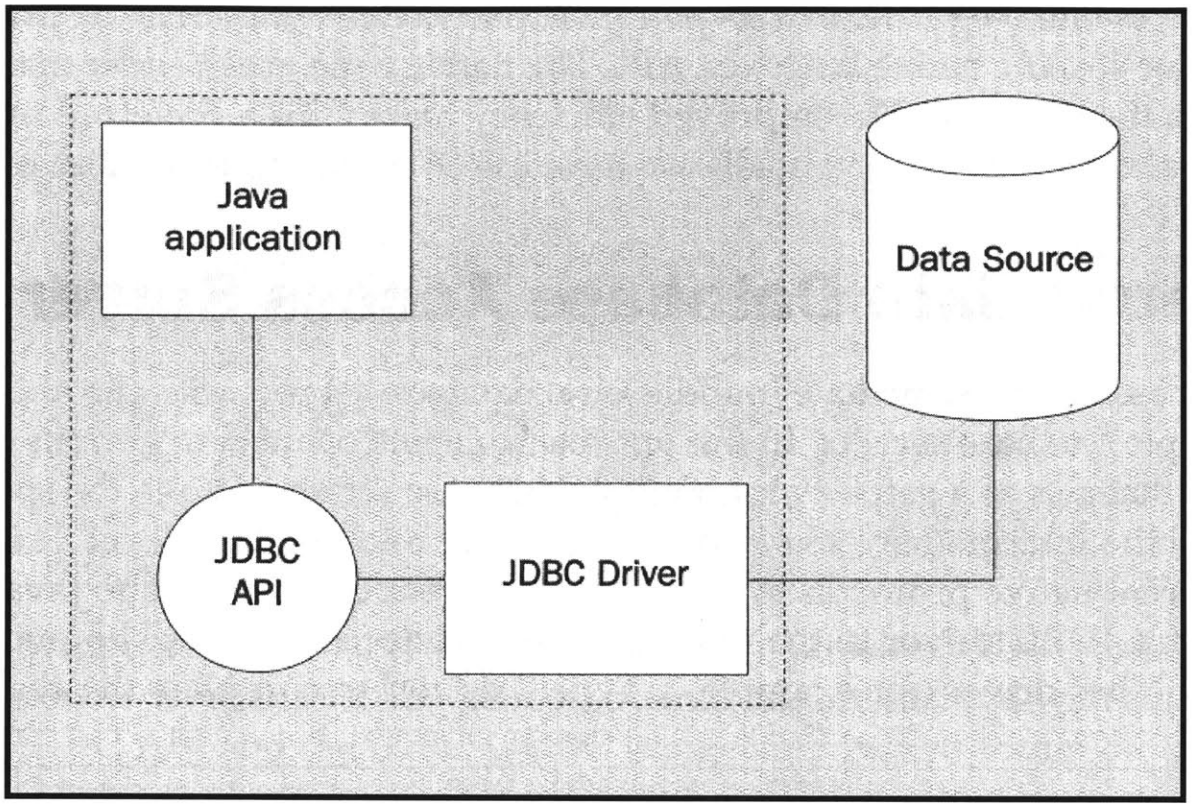

Figure 8: Pure Java JDBC Driver

4.3.2 Loading a Database Driver and making a connection

In the case study project the database used is the MySQL database, which is a freely available open source database. Following are the steps involved in loading and creating a connection to the MySQL database.

- To the load the driver, you have to load the appropriate class; a static block in the class itself automatically makes a driver instance and registers it with the JDBC driver manager. This is done by using the 'Class.forName' method. This method takes a string representing a fully qualified class name (i.e., one that includes package names) and loads the corresponding class. This call could throw a ClassNotFoundException, so it should be inside a try/catch block.

  Example:

```
try{
      Class.forName("org.gjt.mm.mysql.Driver");
  }

catch(ClassNotFoundException cnfe)
{
      System.err.println("Error loading driver: "+ cnfe);
}
```

- Once you have loaded the JDBC driver, you need to specify the location of the database server. URLs referring to databases use the jdbc: protocol and have the server host, port, and the database name embedded within the URL. The exact format is defined in the documentation that comes with the particular driver. Below is the format for MySQL

```
String dbDriver = "org.gjt.mm.mysql.Driver";
String dbURL = "jdbc:mysql://localhost/test_water?user=&password=";
```

- To make the actual network connection, pass the URL, the database username, and the password to the **getConnection** method of the *Driver-Manager*. Below is the format for MySQL

```
Connection dbCon;
dbCon = DriverManager.getConnection(this.getDbURL());
```

4.3.3 Creating & Executing SQL statements

- Once a connection has been established, we need to create a statement object. A statement object is used to send the queries and the commands to the database and is created from the connection as follows.

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

- Once a statement object has been created, you can use it to send SQL queries by using the **executeQuery** method, which returns an object of type **Resultset** (ref sec 4.3.4)

For example:

```
String sql = "SELECT * from MeterRead ";
Resultset rs = statement.executeQuery(sql);
```

- To modify the database, we can use **execUpdate** instead of **executeQuery** and supply a string that uses UPDATE, INSERT or DELETE.

4.3.4 Using the ResultSet object

The simplest way to handle results from the database is to process them one row at a time, using the resultSets **next** method to move through the table a row at a time. Within a row, **Resultset** provides various getXxx methods that take a column name as an argument and return the value as a variety of different Java types.

Below is an example that prints the values of the first 2 columns in all rows in a resultSet.

```
While(resultSet.next())
{
      System.out.println(results.getString(1) + "
" +
                        results.getString(2)) ;
}
```

4.3.5 Closing the Connection

To close the connection to the database explicitly, you would do the following

```
Connection.close()
```

You should postpone this step if you expect to perform additional database operations, since the overhead of opening a connection is usually large.

5 Water Metering Case Study: Applying J2EE Technology Design Principles to the Project

5.1 Introduction

Chapter 3 gave a brief introduction to the case study project. It also discussed the functionalities in brief. Chapter 4 discussed the fundamentals of the Java technologies that are used to run the functionalities (eg: Bill Preparation) in the project. This chapter will discuss how the various Java technologies interact with each other how they were used to implement the functionality in software system that was built for the case study project. It will also explain the design concept (framework) behind the system for the project.

In this chapter, the thesis will attempt to explain the application of the Java technologies by explaining one of the functionalities in the project namely the 'Bill Preparation– Abnormal Increase' (BPAI). It should be noted that all other functionalities in the project follow the same design principles and code structure in implementing them.

5.2 JSP design (Page-Centric)

There are 2 main approaches in JSP design:

1. **Page Centric** or **Client-Server** designs. In these designs, requests are made directly to the JSP page that produces the response.

2. **Dispatcher** or **n-tier** designs, in which the request is originally made to a JSP or a servlet that acts as a mediator or controller- dispatching the requests to JSP pages and javabeans as appropriate.

In this project, the page-centric approach was used, as the number of users is very limited and it is not on a distributed level right now. If the system has to scale to a distributed system with a large number of users executing various transactions simultaneously affecting the database, then probably the Dispatcher or n-tier design would be more appropriate.

In the Page-centric approach, JSP's or servlets access the enterprise resources (a database, for example) directly or through a javabean and generate the response themselves. The advantage of such an approach is that it is simple to program and allows the page author to generate dynamic content easily, based on the request and the state of the resources.

There are two main variants here: the **Page-View** and **Page-View with a Bean.** Both these approaches are used in the case study project as and when appropriate.

5.2.1 Page View

This design approach involves direct request invocations to a JSP page with embedded Java code, and markup tags that dynamically generate output for substitution within the HTML.

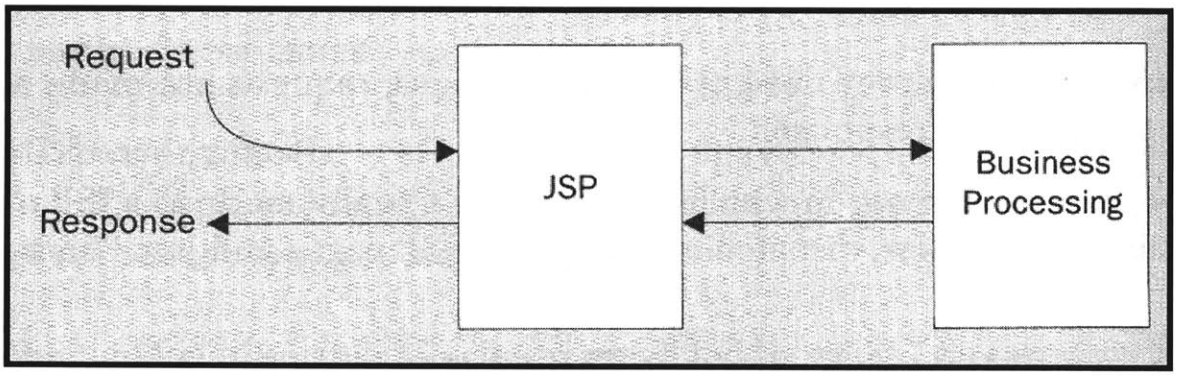

Figure 9: Page View (JSP Design)

This approach has many advantages.

- It is easy and is a low overhead approach from a development standpoint
- All the Java code is embedded within the HTML and so changes are confined to a smaller area, reducing complexity

The main disadvantages are:

- As the scale of the system grows, problems arise such as including too much business logic in the page.
- Differentiation of presentation & content is not really achieved if there is large amount of Java code in the HTML.

5.2.2 Page-View with Bean

This design approach is used when the page-view approach starts embedding large amounts of business logic and data access related code into the JSP. The design becomes more sophisticated as shown in the figure below

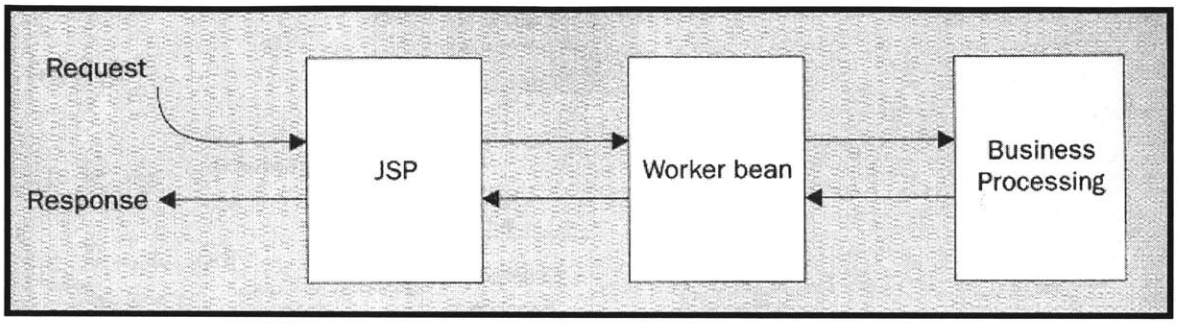

Figure 10: Page View with Bean (JSp Design)

The Java code representing the business logic and simple data storage implementation is put into the javabean and the JSP page accesses all this code through Bean instantiation and method calls. This segregation of business logic code in the javabean leaves the JSP page with minimal code.

The main advantage of this approach is that a distinction can be made in the JSP between the HTML and the Java Code. The functionality of the bean can be owned wholly by the software developer and does not have to worry about it interfering with the design code (HTML), which is handled by the web developer.

This design approach does not scale up to larger distributed systems where you have to tackle issues like authentication, sessions and database connection pooling, load-sharing etc.

5.3 Bill Prep Functionality – Abnormal Increase (BPAI)

The thesis will now discuss the 'Bill Prep Abnormal Increase' (BPAI) functionality in detail to explain how some of the J2EE technologies were used to implement this functionality in the MIT system, that was built for the DPW of Arlington.

5.3.1 Explanation of the Functionality (BPAI)

The BPAI functionality basically is used to figure out which meters had an abnormal increase in usage in the time period specified. Keep in mind that the software runs via the web browser, which is the UI. All the time periods are in 'days'.

The user inputs all the parameters on a web page called the **'Report Configuration'** page of the system. The configuration page is common for all the functionalities in the system. The user views the results for the BPAI by browsing to the 'Bill Prep Results' section on the Homepage. Each of the functionalities in the project has a separate results page where the user could view the results independently.

The BPAI functionality uses 4 parameters as input from the user namely

Report Date (End Date): This is the date on which the user runs all the analyses. The user defines the end date of the analysis at the top of the **Report Configuration** page. All the Analyses in the system take this date as the end date for the respective analysis.

Current Time Period: The current time variable defines the first of two time periods needed for the calculations. The system defines the start date of the time period by subtracting the current time value from the end date of the analysis.

Past Time Period: The past time variable defines the second of the two time periods. The past time variable defines the start and end date of the past time period. The end of the past time period will be the day before the start of the current time period. The system will then calculate the start date of the past time period by subtracting the past time variable from the start date of the past time period.

Percent Increase: The percent increase parameter allows the analysis to flag suspect meters when the percent increase between average daily use in the current time period and the average daily use in the past time period is greater or equal to the percent increase parameter. Only flagged meters will appear in the results report.

5.4 Javabean Usage in BPAI (Modularity)

In every software system there are certain tasks or functions that will be performed repetitively and it is implicit that you should not have to rewrite the code for them wherever you use them. For eg. Opening a database connection, writing database query output into a table in HTML etc.

In the case study project there were certain classes that were coded so that they take care of these repetitive functions or tasks. For the entire class diagram please refer to the Master of Engineering thesis titled *'Architecture of Near Real-Time Monitoring Systems for Water Distribution Systems'* by Sebastian Bogerhausen, year 2001 (Reference no. 9). This thesis will discuss only the methodology and class hierarchy required to implement the BPAI functionality.

The BPAI functionality uses the Page View with Bean design approach. The javabean, which is specific to the BPAI functionality is the **BPIncrease** bean. The functionality also uses other general-purpose beans that are used throughout the project code for repetitive tasks. These are the **DbBean** and **ReportBean**. For detailed explanation of these javabeans, please refer to the above-mentioned thesis (Reference no. 9). In brief these beans do the following:

BPIncrease bean:

This bean contains the business logic for the BPAI functionality in it. It has a **setParam** method, which is called when you want to set the parameters for this functionality. It also has a **run** method, which actually does all the calculations and writes the results to a table in the database.

DbBean bean:

This bean is a general-purpose bean, which is used for all database activities. These activities include connecting, querying, deleting, updating and disconnecting form a database.

ReportBean:

This bean is used for taking all the parameters on the **Report Configuration** page and writing them to a table in the database. All the functionalities including BPAI use this bean to store the parameters that were set for the analysis in the respective table in the database.

5.5 Database Table Structure

For all the functionalities including BPAI, the tables in the database have a logical structure. Basically, all the functionalities have a table for storing the parameters that are set on the **Report Configuration** page of the system. All the functionalities and sub functionalities have a 'results' table, which stores the results of the analysis that was carried out on the data in the respective functionality. For more details on the database design and the data model please refer the Master of Engineering thesis titled ***"The Business Transformation Effects of Information Technology"***, by Bradford Butler, year 2001.

The following tables were used for the BPAI functionality.

Config :

The **Config** table stores all the parameters for all the functionalities that are set for the various analyses to run.

BPIncResults:

The **BPIncResults** stores all the results of the BPAI analysis in it. This table is used to extract the results of the analysis and display it to the user.

5.6 BPAI Functionality Process & Code Explanation

The thesis will now discuss the BPAI working step-by-step giving the user perspective (what the user does and sees) along with the code perspective (what happens behind the scene in the system). All the web pages that are referred below are JSP pages. For the entire code for all pages and javabeans, refer **Appendix A**

Step 0

The user is at the home page of the system and clicks on the **Report Configuration** link.

The screenshot of the system below shows the Home page at the beginning of Step 0

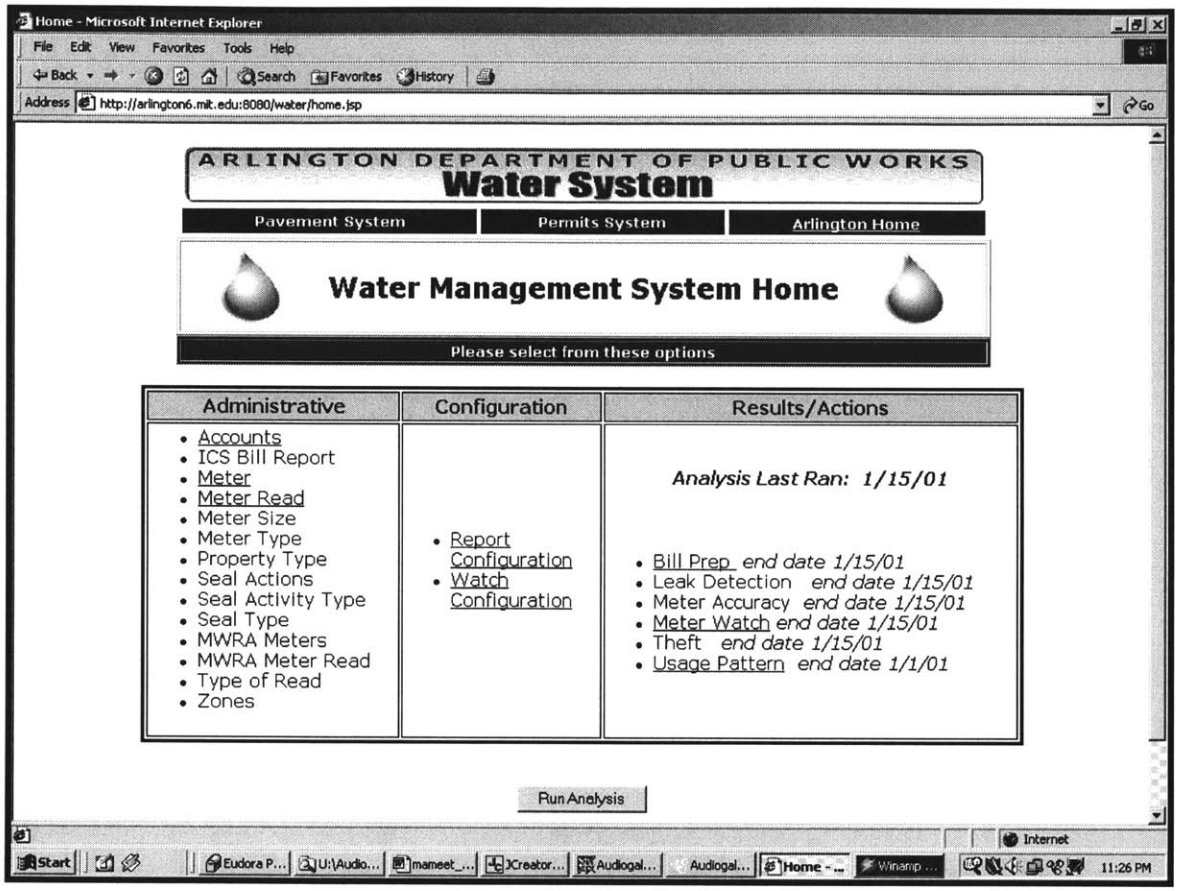

Figure 11: Case Study, Home Page (Screenshot)

Step 1

User Perspective

The user has to input 4 parameters (explained above) to run this analysis. All parameters that have to be entered for any of the analysis have to be entered on the **Report Configuration** page of the system.

Code Perspective

The user inputs the parameters into a HTML form on the **Report Configuration** page.

The screenshot of the system below shows the configuration page at the beginning of Step 1

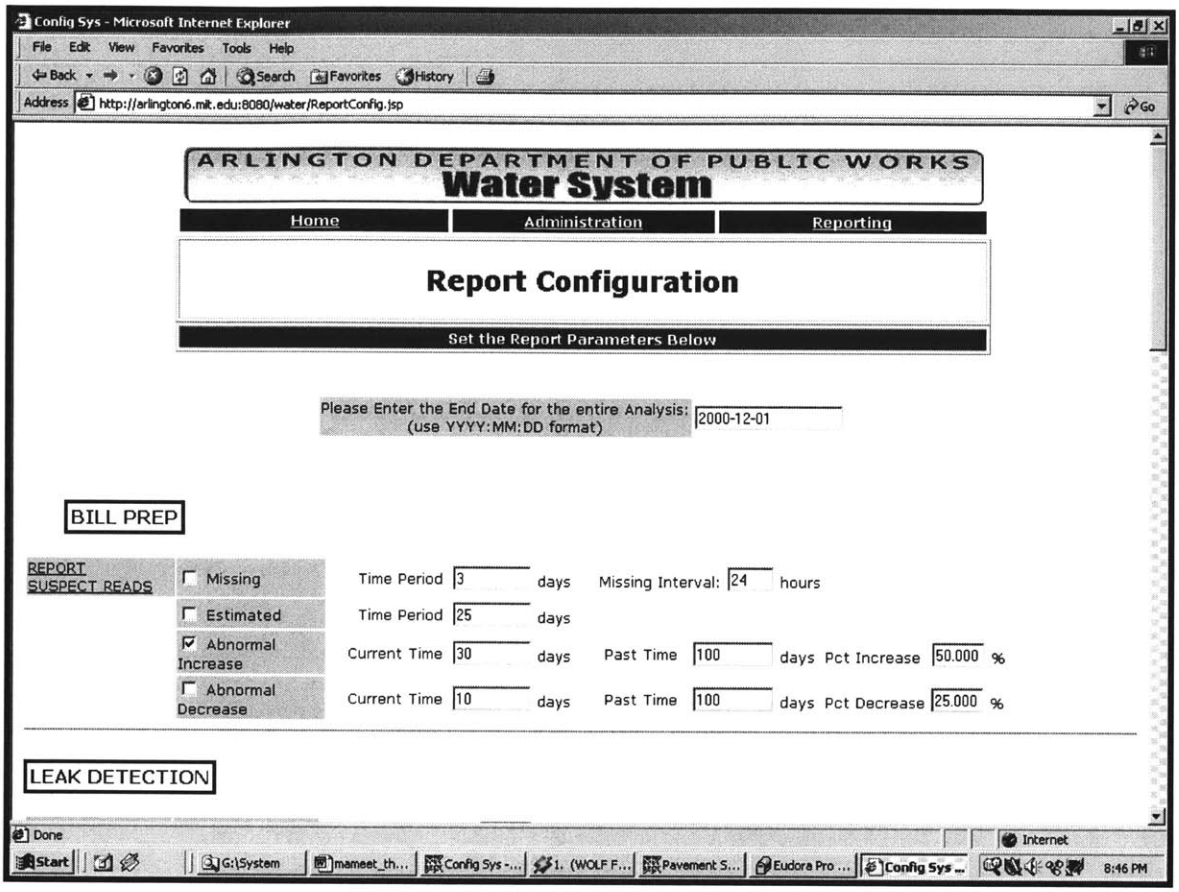

Figure 12: Case Study, Report Configuration page (Screenshot)

Step 2

User Perspective

The user submits the configuration page parameters by pressing the submit button on the **Report Configuration** page. On pressing the submit button the user is taken back to the home page.

Code Perspective

When the user presses the submit button on the **Report Configuration** page, all the parameters are passed to the **ReportBean** which in turn writes the parameters to the **Config** table in the database via the use of the **DbBean** bean. The system then forwards the user to the home page.

Step 3

User Perspective

To run the analysis the user has to press the **Run Analysis** page on the home page. On pressing the 'Run Analysis' button the system brings up an **intermediate page**, which shows the user the various analyses (functionalities) that were selected for processing. The system runs the analysis when a 'Continue' button is pressed on this **intermediate page**. After some processing, the user is forwarded to the home page where he can view the results.

In this analysis, the system compares the average daily use in the current time period to the average daily use in the past time period. If the percent increase in water usage for a meter is greater than or equal to Percent Increase parameter then the system will keep track of it and write it to the results.

Code Perspective

When the user presses 'Run Analysis' button on the table the system accesses the **Config** table in the database for the analyses that were selected. It then shows the user via an **intermediate page** the analyses that will be run. When the user presses the 'Continue' button on the intermediate page, the system instantiates the **BPIncrease** bean and passes the parameters that were set for the analysis to the bean via the **setParam** method in it. Once the bean has all the parameters, the **run** method is called which contains the business logic that runs the analysis. The **run** method also writes the results to the **BPIncResults** table in the database using the **DbBean** bean. After all the analyses are run, the user is forwarded by the system to the homepage.

The screenshot of the system below shows the Home page at the beginning of Step 3

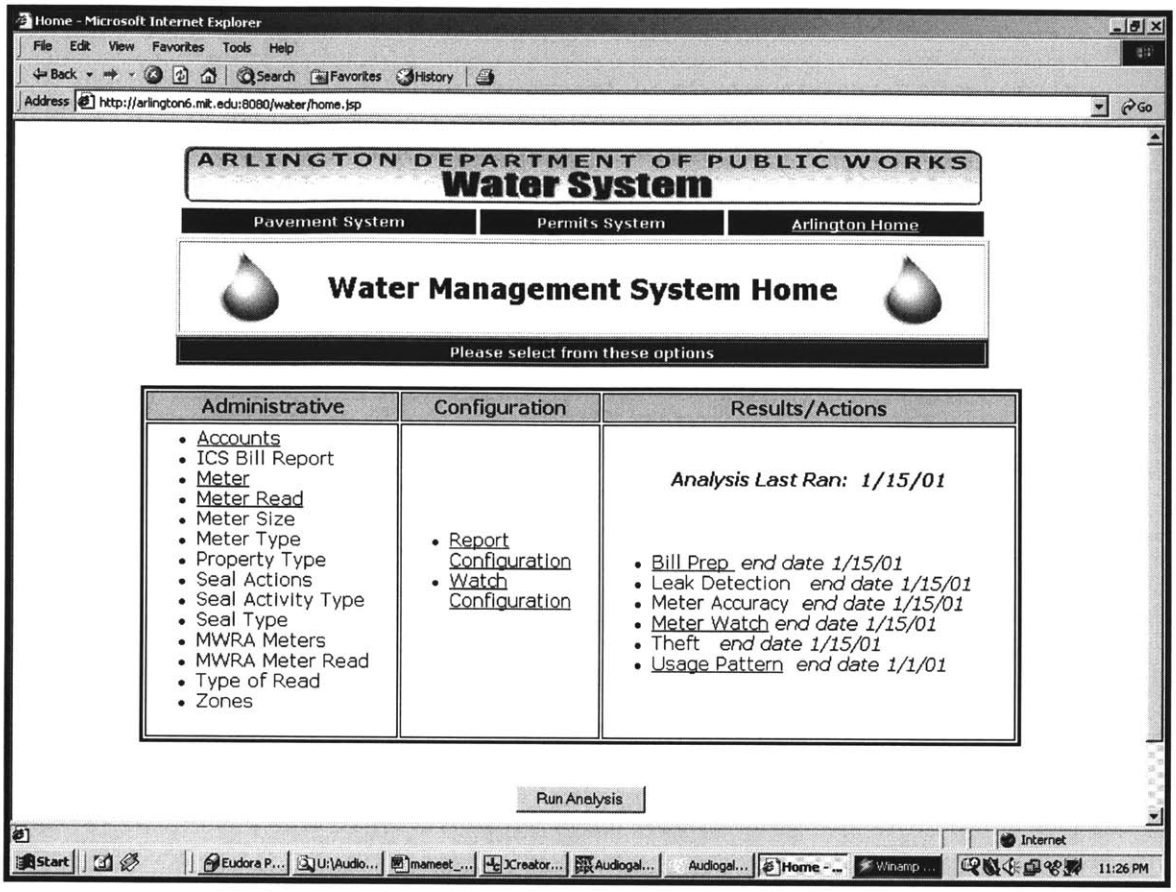

Figure 13: Case Study, Home Page (2) (Screenshot)

The Intermediate page is shown below:

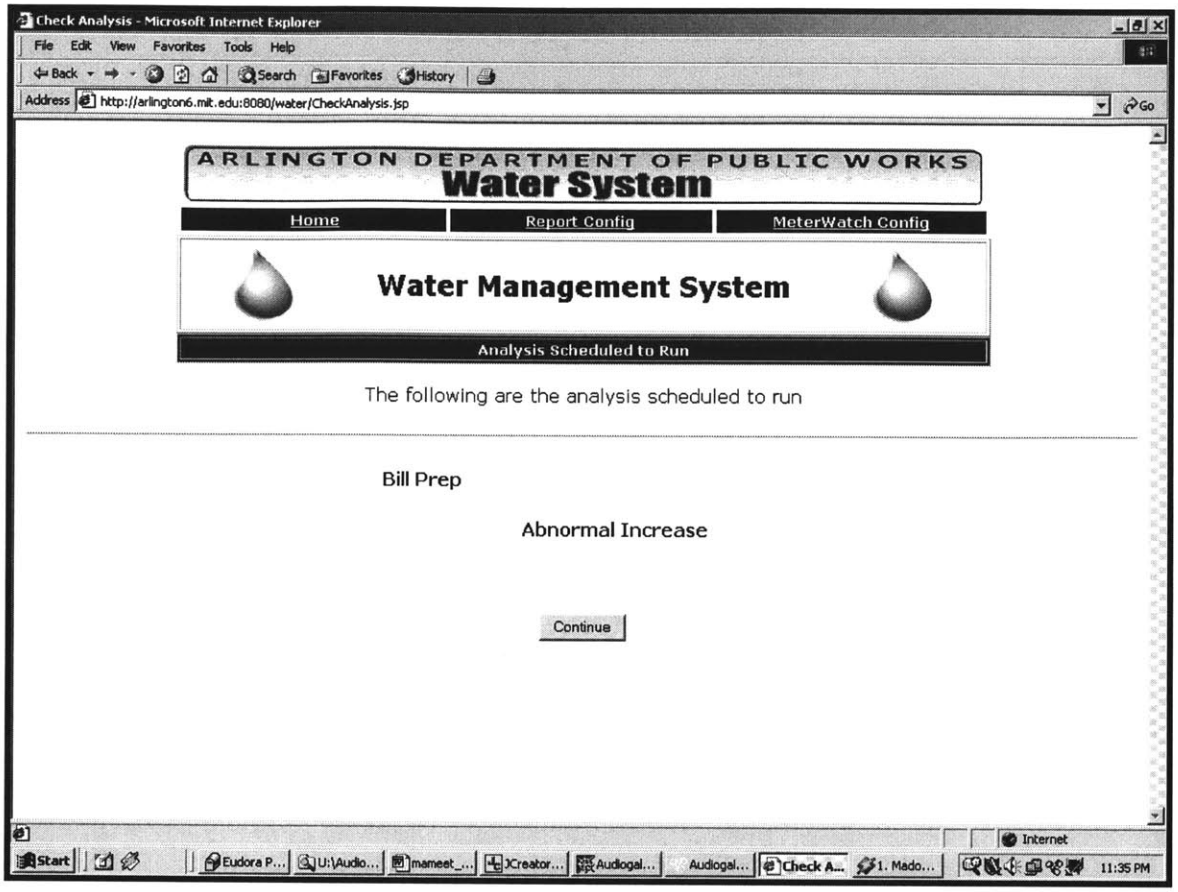

Figure 14: Case Study, Intermediate Page (Screenshot)

Step 4

User Perspective

The user can view the results by browsing to the Bill Prep results link on the home page. When the user clicks on the link the **BillPrepResults** page showing the result is displayed.

Code Perspective

When the user clicks on the 'BillPrepResults' link on the home page, the system queries for the **BPIncResults** table in the database via the **DbBean** bean. Again with the use of the **DbBean** bean the system displays the results of the query in HTML on the browser on the **BillPrepResults** page.

The **BillPrepResults** page is shown below:

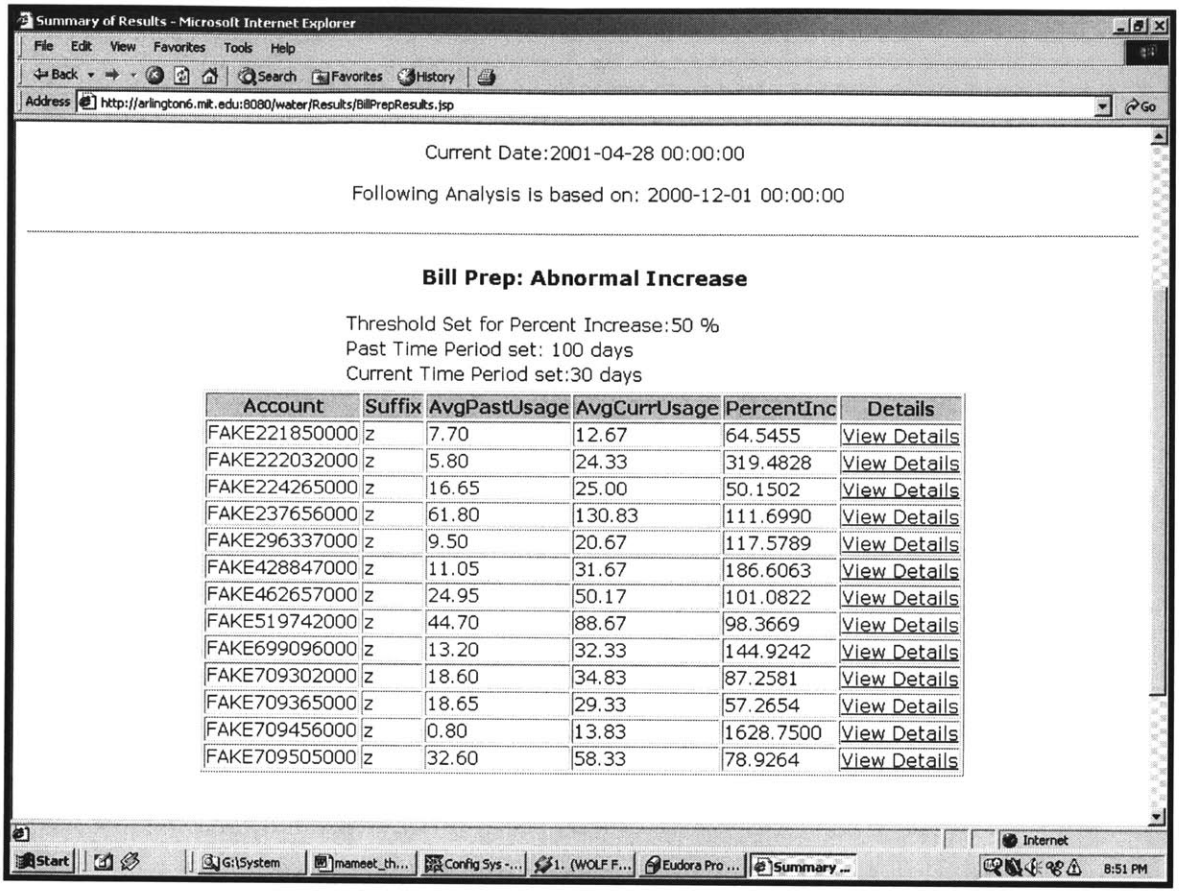

Figure 15: Case Study, Bill Prep Results page (Screenshot)

Step 6

User Perspective

If the user wishes to see details of a particular meter, he can click on the 'View Details' link (shown above) in the results table. The readings for the meter in the specified time period would be then brought up in Excel on the browser, which he can manually graph with minimal effort.

Code Perspective

If the user clicks on the 'View Details' link the system sends information of that particular meter to the **BpAbIncGraph** page, which basically retrieves all the information from the **BPIncResults** in the database. The table of readings is actually output in HTML. But, if

the content type for the particular JSP page (**BpAbIncGraph**) is set to an Excel format then, since Excel 2000 is able to read HTML tables, the entire page opens up in Excel. The user can then use Excel functionality to graph the readings.

Both screens are shown below:

The Excel spreadsheet

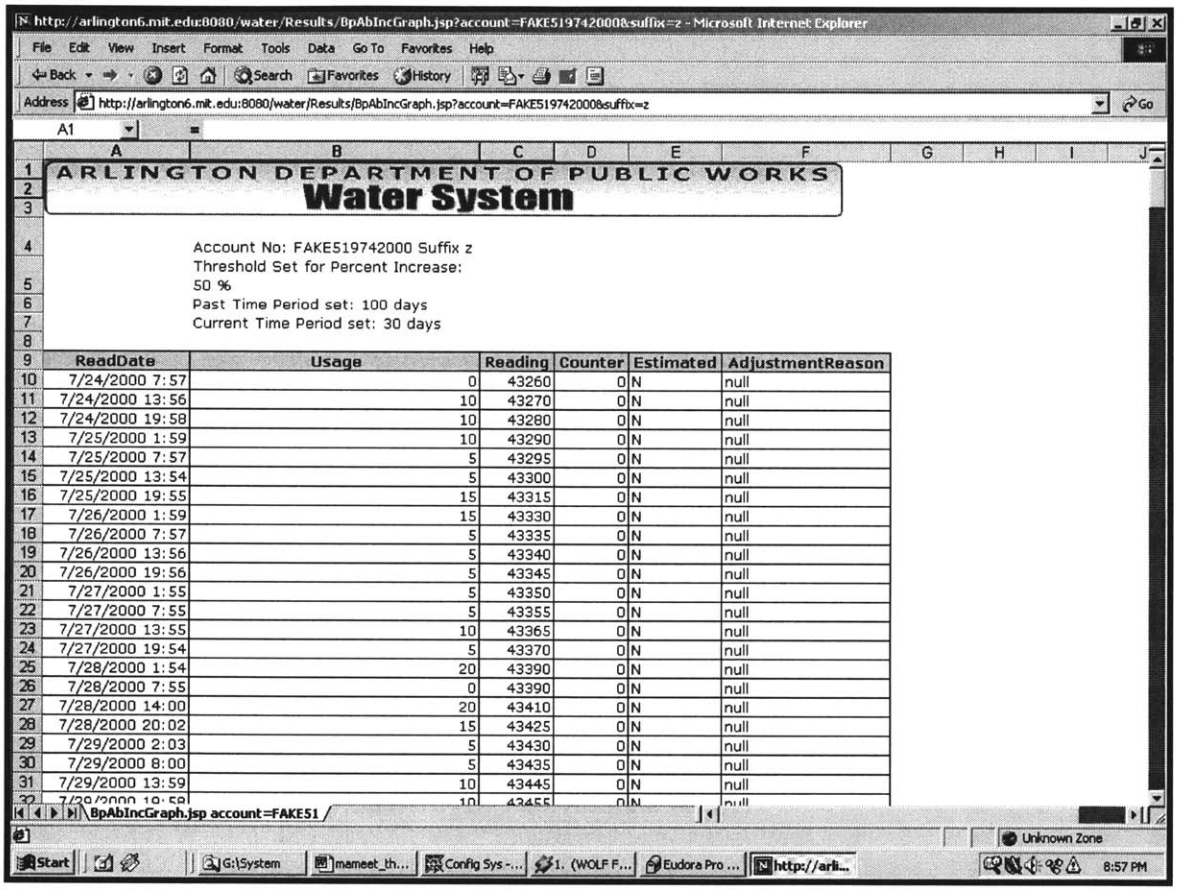

Figure 16: Case Study, Bill Prep Details Page, (Screenshot)

The Excel graph for the above readings is shown below:

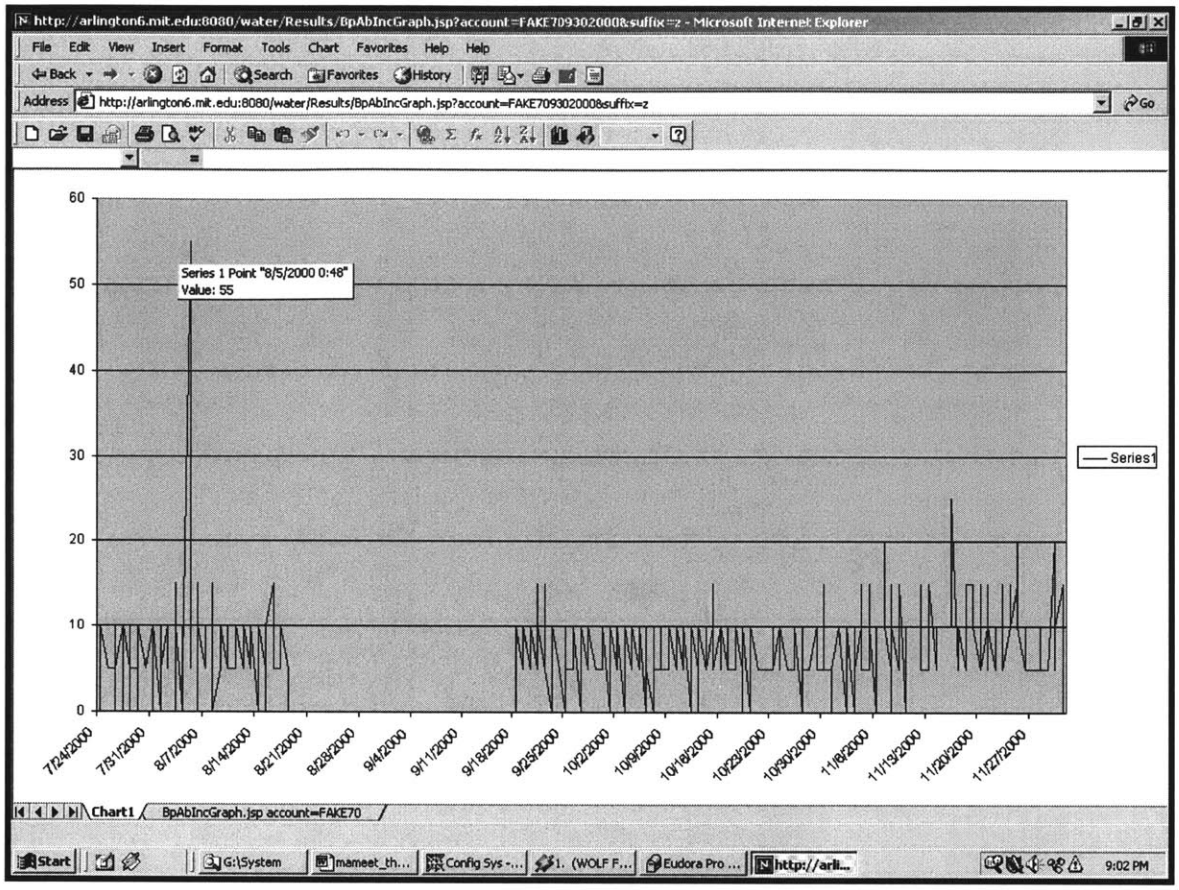

Figure 17: Case Study, BPAI Graph page (Screenshot)

5.7 Current System Trade-Offs

The system is currently is multi-user, but it does not keep track of sessions etc of the people using it. Because of time limitations and scope of project, security issues were not really resolved. Currently, the system could be accessed by anyone in the DPW on the intranet.

It is not designed for distributed computing. For example, if there are 2 users using the same functionality and both users add a new 'Account' to the database via the system, the system cannot track who added the record.

Because of the limitations of the MySQL database, which does not support referential integrity, deleting certain records does not guarantee a cascading delete.

5.8 Future Extensions & Guidelines for a Distributed Computing Environment

- Security functionalities using Login/Password etc would help restricted access.
- Migrating to a better database system other than MySQL because of certain limitations or use of newer versions of MySQL when they are released.
- Use of distributed computing techniques using RMI API etc if the number of users rises drastically.
- The 'Dispatcher' JSP design approach would probably have to be used for Distributed computing using RMI.

6 Alternative to J2EE .NET vs J2EE A Technical Comparison

6.1 Introduction

Sun unveiled the J2EE (Java 2 platform Enterprise edition) for developing multi – tiered enterprise applications. Microsoft not to be left behind came out with the .NET framework. Conceptually, both platforms (or frameworks) are the same: the only difference being that .NET is built around Microsoft platforms and brings in a tight integration amongst them. Having discussed the J2EE platform and its component-based architecture it is appropriate to investigate the components of the .NET platform and compare the two frameworks.

6.2 Components of the .NET platform

Note: * At the time this thesis was written, the .NET framework was at an early stage in its lifecycle and the deep details were still being eked out by the Microsoft team. So most of the "tense" in the paragraphs is futuristic.

The five pillars for the .NET platform are:

- **.NET Framework and Microsoft Visual Studio.NET developer tools**
  This will provide the development environments for the applications to be written.
- **Windows and the Microsoft .NET Enterprise Servers**
  Severs that will connect the applications and the development environment to the Internet.
- **.NET Foundation Services**
  This will provide basic services to the users and developers like identity. The coherent identity across services will allow applications from exchanging meaningful data relevant to users and developers.
- **Device software**
  This layer would be to create the platform independence capability that java has. The .NET framework should ideally work on any device provided it is connected to the Internet.

These four together will form the .NET platform. But, what Microsoft believes will provide it the advantage is the Microsoft compatible applications.

- **.NET - compatible applications**

  This will consist of a set of applications that will run on top of the .NET platform like Microsoft's existing products, the MSN network and office tools etc. Suppose an enterprise application built using .NET platform needs data analysis, then this can be done using Microsoft Excel.

6.3 Comparing .NET with J2EE

The two technologies can be compared at the technical component level (item-by-item) and at the practical implementation level.

6.3.1 Technical Component Level Comparison

	Microsoft .NET	J2EE
Programming Language	Based on C# programming language	Based on Java programming language
Pre-written codes	.NET common components	Java core API (application program interface)
Internet Applications	Active Server Pages+ (ASP+)	Java ServerPages (JSP)
Common language for runtime	Internal Language (IL)	Java Virtual Machine and CORBA (common object ORB architecture), IDL (interface definition language) and ORB (object request broker)
User interface component framework	Win Forms and Web Forms	Java Swing
Database connectivity	ADO (Activex data object) and SOAP (simple object access protocol) -based Web Services	JDBC (Java database connectivity), EJB enterprise java beans), JMS (Java message service) and Java XML Libraries (XML4J, JAXP)

Table 1: Technical Component Level Difference between .NET & J2EE

Key differences between the components described above:

Programming Language: C# and Java both derive from C and C++. Most significant features (e.g. garbage collection, hierarchical namespaces) are present in both. Java runs on any platform with a Java VM. C# runs only on Windows in the foreseeable future.

C# code always runs natively. Java code typically runs as interpreted bytecodes, and can run natively. C# is either compiled entirely to native code, or it is compiled into the common language runtime bytecodes and then just-in-time compiled to native code during execution. Java code, on the other hand, typically runs as runtime-interpreted bytecodes (from which its cross-platform abilities spring), and can also run in a just-in-time compiled context. Some Java native-code compilers also exist (Jove, BulletTrain, JET, etc.).

Internet Applications: ASP+ will use Visual Basic, C#, and possibly other languages for code snippets. All get compiled into native code through the 'Common Language Runtime' (CLR, see below) as opposed to being interpreted each time, like ASPs. JSPs use Java code (snippets or Javabean references), compiled into Java bytecodes.

Common language for runtime: .NET 'Common Language Runtime' (CLR) allows code in multiple languages to use a shared set of components, on Windows. Code and objects written in one language can, ostensibly, be compiled into the IL runtime, once an IL compiler is developed for the language. The CLR underlies nearly all of .NET framework (common components, ASP+, etc.). Java's Virtual Machine spec allows Java bytecodes to run on any platform with a compliant JVM.

User interface Component framework: Win Forms and Web Forms RAD development is supported through the MS Visual Studio IDE (Integrated Development Environment) - no other IDE support announced presently*. Swing support available in many Java IDEs and tools.

Database connectivity: ADO+ is built on the premise of XML data interchange (between remote data objects and layers of multi-tier apps) on top of HTTP (AKA, SOAP). .NET's web services in general assume SOAP messaging models. EJB, JDBC, etc. leave the data interchange protocol at the developer's discretion, and operate on top of either HTTP, RMI or IIOP.

6.3.2 Practical Implementation Differences (Critical differences)

There are critical differences between the two frameworks at a number of levels.

Dot-NET and J2EE both stand on a foundation of programming languages, object models and virtual machines. The most striking difference between the two frameworks is the design goals of their runtime environment and how these support very different programming and deployment schemes. The cliché developing in the development community around this point is that "Java is language-specific and platform-independent, and dot-NET is language-independent and platform-specific." This cliché is in many ways an oversimplification since numerous J2EE implementations aren't entirely cross-platform. Microsoft has made some preliminary attempts to make dot-NET cross-platform, such as providing the CLR specification and the C# language as well as announcing versions of the CLR for a variety of non-PC devices, including the Linux platform. The cliché remains fairly accurate, though, because the J2EE specification is fundamentally cross-platform and the full dot-NET story incorporates Windows as a central piece.

J2EE is a framework that evolved out of the core Java environment, which is founded on the Java language, the Java Virtual Machine (JVM) and the Java core APIs. The Java programming model calls for class descriptions written in Java to be compiled into platform-independent bytecodes as defined by the 'Java Virtual Machine' specification (see figure below).

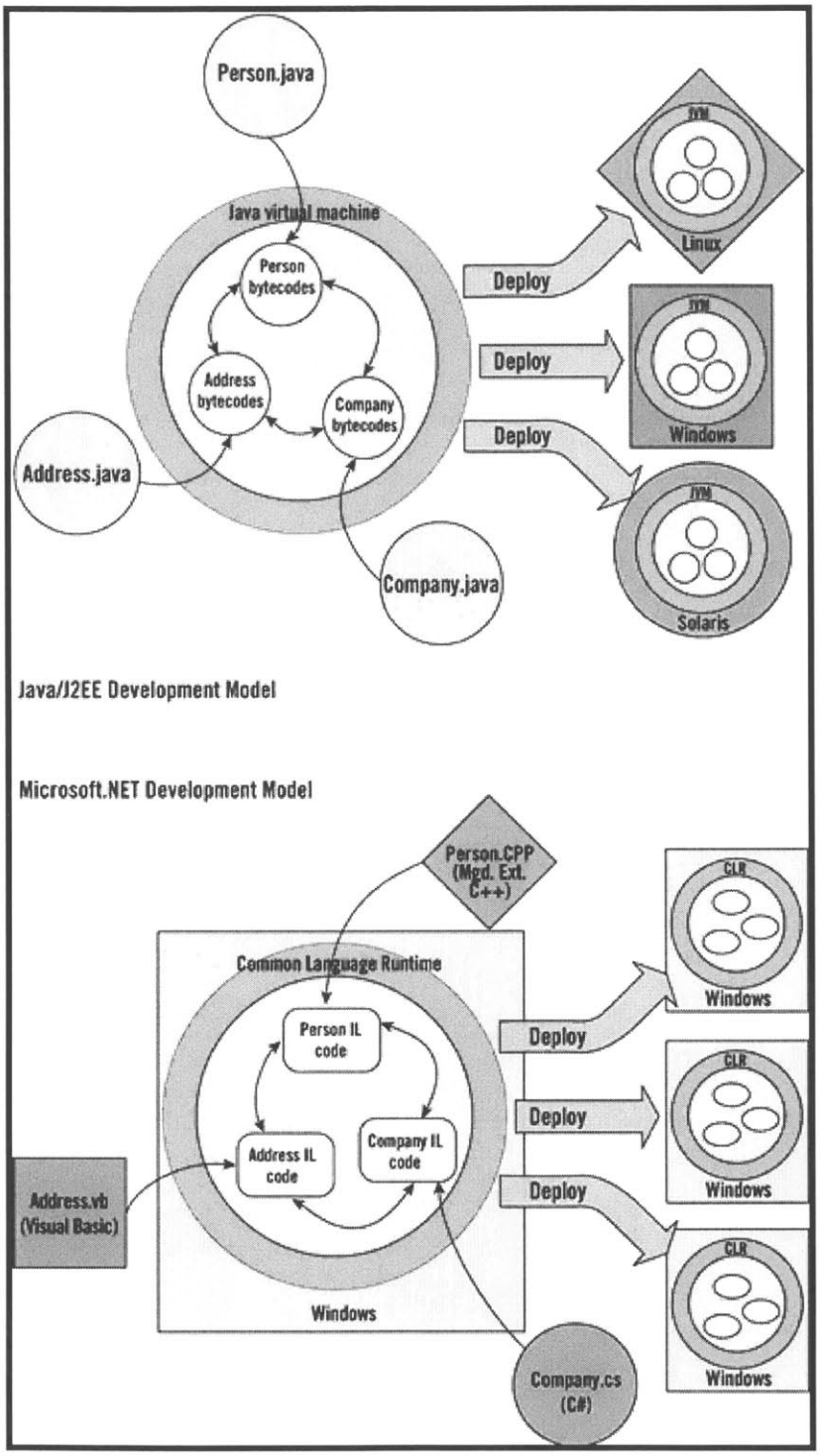

Figure 18: J2EE vs .NET Architecture

These bytecodes are then interpreted and executed by an implementation of the JVM targeted for the particular platform at hand—Solaris, Windows, Linux and so on. The JVM defines and provides a standard runtime environment that provides memory management and security. Alternatively, Java code (or the resulting Java bytecodes) can be compiled to a native executable for a specific platform, or runtime compiled to native code by a just-in-time (JIT) compiler. So, development in J2EE is meant to be done in Java, resulting in class bytecodes that can be run on any platform with a JVM or a Java-to-native compiler. Access to objects and components written in other languages is possible, using the Java Native Interface (JNI) and CORBA, but these are largely ancillary to the core development and runtime model. In both cases, these are also rather complicated APIs, which in part hinders their broad adoption.

Microsoft's dot-NET framework is based on its 'Common Language Runtime' (CLR), which is composed of a specification for language-independent intermediate language (IL) code and a runtime that provides memory management, security and so on. This may seem analogous to the Java runtime architecture, but the difference is that code targeted for the dot-NET framework can be written in any language that supports the CLR's core component model. A compiler must be developed which compiles the code into IL—once there, the objects and components can run side-by-side and interact with components written in other languages and compiled into the CLR. Blocks of IL code are JIT-compiled into native Windows code, or your whole application and assembly can be compiled once into a native Windows DLL or EXE as shown in Figure 1.

The interesting thing about this difference in the two frameworks, and the reason it's a critical differentiator, is that this relatively low-level design feature strikes right at each framework's strategic direction. Java is designed to be a 'unified programming model' that is platform- independent. Dot-NET, on the other hand, is a 'unified platform' (Windows) that is language independent and deeply XML-enabled.

The language and platform distinction is critical.

Java's true potential is realized by a development team and its surrounding organization, only when a critical mass of development (or all of it) is done in Java. If, for some reason (legacy system, third-party requirements, component availability), a subset of a given

system falls outside of the Java environment, things get complicated, and you have to turn to CORBA or JNI or other ways to bridge the gap.

This language-centric approach is also limiting strategically. It's easy to imagine some powerful improvements to the Java core environment that would be best accomplished by making changes to the syntax of the Java language, and these changes are very difficult to make because there is a vertical dependence, from the JVM to the J2EE application services, on the fact that everything happens in Java bytecodes.

Microsoft dot-NET, on the other hand, realizes its true potential only if it is necessary and prudent to create and manage components developed in multiple languages, and do it all on Windows. On the face of it, being able to use the language of your choice (from the set that supports MSIL compilation, that is) seems like a valuable possibility—if I don't know C# or C++ or Java, I can stay productive and write code in what I know, instead of starting a new learning curve. This is appealing for both developers and managers alike, but the CLR achieves somewhat less than this in reality.

You can't write standard, ANSI-compliant C++ or standard COBOL and make it work in the CLR sandbox. To make your code accessible to other CLR components, you have to write your code in Managed Extensions to C++ or COBOL#, which are syntactic variants of the original languages that they share names with. These variants include syntax extensions that are required in order to provide all of the component meta-data that the CLR requires, which can't be supported by the standard language. This means two things: there is still a learning curve to be climbed (although probably not a large one) in order to program to dot-NET's CLR, and the code that you write for dot-NET won't be understood by the standard compilers or interpreters for these languages.

6.4 Conclusion

At the highest level, it is logical to perceive that Microsoft is fixing its Web application environment, making it a peer (at the very least) to J2EE, so when dot-NET arrives we'll have two very capable development frameworks to choose from.

One has to decide which framework to adopt. Given all the technology issues we discussed, the path to a decision is fairly simple. In any significant development effort,

whether you're a consultant on a custom development engagement, an internal developer of Application Service Provider (ASP) offerings, or building the internal enterprise infrastructure for an organization, you need to decide who your target customers are (today and in the future), what platforms you need to support from a strategic level (today and in the future), and what framework provider you want to partner with to push forward (today and in the future).

If an organization has the luxury of only supporting the Windows platform (with respect to both customers and internal development needs), and if it feels Microsoft is the right partner (for development tools, servers, administrative systems and so on) for the organization, then you have an easy decision indeed. Alternatively, if you have the luxury of defining a Web development practice based on a single language, and can't say with finality that Microsoft and Windows can be your sole vendor and platform then, you have an equally easy decision to make.

Conversely, if you face the strategic need to support (or standardize on) Unix and other non-Windows servers, or if you want to leave your options open in other areas (tool vendors, application servers, component vendors and so on), then J2EE is an appropriate choice. Alternatively, if supporting development in multiple languages is a strategic need of yours, and you can live with a single platform and tool vendor, then dot-NET might be a better choice.

Of course, very few groups or organizations find themselves in any of these neat compartments, but the final decision lies with assessing what are the needs of the software application in terms of its architecture and also the purpose for which it is being built in the first place.

7 References:

1. Marty Hall, ***Core Servlets and Java Server Pages***, Sun Microsystems Press, A prentice Hall Title

2. ***Professional Java Server Programming***, J2EE Edition, Wrox Press Ltd. ®

3. B.Butler, S.Bogershausen, M.Khanolkar, ***Master of Engineering Project Report, Arlington Water Project***, MIT ®

4. Jim Farley, Online Article: ***Microsoft .NET vs. J2EE: How Do They Stack Up?***, March 2001, www.oreilly.com ,

5. Jim Farley, Online Article: ***Picking a Winner: .NET vs. J2EE***, March 2001, Software Development magazine online, www.smagazine.com

6. Equitymaster.com, Online Article: ***World war III: .NET vs J2EE,*** Feb 2001, www.yahoo.com

7. Naughton Schildt, ***The Complete Reference, Java 2, 3rd edition***, McGraw Hill

8. Bradford Butler, ***The Business Transformation Effects of Information Technolog,*** Massachusetts Institute of Technology, Master of Engineering Thesis, 2001

9. Sebastian Bogershausen, ***Architecture of Near Real-Time Monitoring Systems for Water Distribution Systems***, Massachusetts Institute of Technology, Master of Engineering Thesis, 2001

8 Appendix A

This Appendix contains the code for the various components that help implement BPAI. The code that is highlighted in bold is the more useful from the reader's perspective. The code, which implements the BPAI is given here in the Appendix. This piece of code is after the configuration parameters are set in the Report Configuration page

8.1 Javabean code that was used in the project to run general tasks with the database

8.1.1 DbBean.java

This is the javabean that is used for all database actions like connecting to the database, querying, throwing out results from the database in HTML, closing the database connection etc.

```
package Water;

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

public class DbBean
{
        Connection dbCon;
        String dbURL =
"jdbc:mysql://localhost/test_water?user=&password=";
        String dbDriver = "org.gjt.mm.mysql.Driver";

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

        public Connection getDbBeanConnection()
        {
            return dbCon;
        }

        public void close() throws SQLException
        {
                dbCon.close();
        }

        public void execAdd(String sql) throws SQLException
```

```
        {
                Statement st = dbCon.createStatement();
                st.executeUpdate(sql);
        }

            public void execUpdate(String sql) throws SQLException
        {
                Statement st = dbCon.createStatement();
                st.executeUpdate(sql);
        }

            public void execDelete(String sql) throws SQLException
        {
                Statement st = dbCon.createStatement();
                st.executeUpdate(sql);
        }

        public DBResults getQueryResults(String query,
                                                    boolean close)
        {

            try
            {
                        DatabaseMetaData dbMetaData =
dbCon.getMetaData();
                        String productName =
                    dbMetaData.getDatabaseProductName();
                        String productVersion =
                    dbMetaData.getDatabaseProductVersion();
                        Statement statement = dbCon.createStatement();
                        ResultSet resultSet =
statement.executeQuery(query);
                        ResultSetMetaData resultsMetaData =
                    resultSet.getMetaData();
                        int columnCount =
resultsMetaData.getColumnCount();
                        String[] columnNames = new String[columnCount];
                        // Column index starts at 1 (a la SQL) not 0 (a
la Java).
                        for(int i=1; i<columnCount+1; i++)
                        {
                        columnNames[i-1] =
                        resultsMetaData.getColumnName(i).trim();
                        }
                        DBResults dbResults = new DBResults(dbCon,
productName, productVersion,

columnCount, columnNames);
                        while(resultSet.next())
                        {
                                String[] row = new String[columnCount];
                    // Again, ResultSet index starts at 1, not 0.
                        for(int i=1; i<columnCount+1; i++)
                        {
                                String entry = resultSet.getString(i);
                                if (entry != null)
                                {
```

```
                                    entry = entry.trim();
                                }
                                row[i-1] = entry;
                            }
                            dbResults.addRow(row);
                        }
                            if (close)
                            {
                                dbCon.close();
                            }
                            return(dbResults);
                    }
                    catch(SQLException sqle)
                    {
                            System.err.println("Error connecting: " +
sqle);
                            return(null);
                    }
                    }

            public ResultSet execResults(String sql) throws SQLException
            {
                    Statement st = dbCon.createStatement();
                    ResultSet rs = st.executeQuery(sql);
                    return (rs==null) ? null : rs;
            }

            public String getDbDriver()
            {
                    return this.dbDriver;
            }

            public void setDbDriver(String newValue)
            {
                    this.dbDriver = newValue;
            }

            public String getDbURL()
            {
                    return this.dbURL;
            }

            public void setDbURL(String newValue)
            {
                    this.dbURL = newValue;
                }
}
```

8.2 Code which implements the Analysis for Bill Preparation Abnormal Increase

8.2.1 Run Analysis.jsp

This page actually initiates the analysis after the 'continue' button is pressed on the intermediate page that show which analyses are scheduled to run.

```
<%@ page language="java" import= "java.sql.*, java.lang.*, Water.*"%>
<html>

<head>
<title>Run Analysis</title>

<link rel="stylesheet" type="text/css"
href="arlington_style_sheet.css">
</head>

<body>
<div align="center">
  <center>
  <table border="0" cellspacing="4" cellpadding="2" width="722">
    <tr>
      <td colspan="3" width="708">
        <p align="center"><img border="0" src="banner_water.gif"
width="708" height="49"></td>
    </tr>
    <tr>
      <td bgcolor="#000080" align="center" width="240"><b><a
href="home.jsp"><font face="Verdana" size="2"
color="#FFFFFF">Home</font></a></b></td>
      <td bgcolor="#000080" align="center" width="240"><b><a
href="ReportConfig.htm"><font face="Verdana" size="2"
color="#FFFFFF">Report
        Config</font></a></b></td>
      <td bgcolor="#000080" align="center" width="241"><b><a
href="MeterWatch/MeterWatch.htm"><font face="Verdana" size="2"
color="#FFFFFF">MeterWatch
        Config</font></a></b></td>
    </tr>
  </table>
  </center>
</div>
<div align="center">
  <table border="1" width="722" bgcolor="#FFFFFF">
    <tr>
      <td width="100%">
        <div align="center">
          <center>
          <table border="0" width="650">
            <tr>
              <td>
                <p align="center"><img border="0" src="drop.gif"
width="57" height="71"></p>
              </td>
              <td>
```

```
                <p align="center"><font face="Verdana" color="#000080"
size="5"><b>Water
                Management System</b></font></td>
              <td>
                <p align="center"><img border="0" src="drop.gif"
width="57" height="71"></p>
              </td>
            </tr>
          </table>
          </center>
        </div>
      </td>
    </tr>
  </table>
</div>
<div align="center">
  <center>
<table border="1" width="722" bgcolor="#800000">
  <tr>
    <td width="100%">
      <p align="center"><font color="#FFFFFF" face="Verdana"
size="2"><b>Running Analysis</b></font></td>
  </tr>
</table>
  </center>
</div>
<p align="center">Processing ...</p>
<hr>


<%
      //check which analyses are scheduled to run by querying the
Config table
      ResultSet rs;
      String sql = "SELECT
EndDate,BoolBpMiss,BoolBpEst,BoolBpAbInc,BoolBpAbDec,BoolUsgMtSz,"+

"BoolUsgAcctTyp,BoolMtWatch,BpMissDays,"+

"BpMissInterval,BpEstDays,BpIncCurrDays,BpIncPastDays,"+

"BpAbIncPct,BpDecCurrDays,BpDecPastDays,BpAbDecPct FROM Config";
      String result;
      String BgColor = "#C0C0C0";
      String BPMiss, BPEst, BPInc, BPDec, UsgMtSz, UsgAcct,
MtWatch,EndDate;
      String
BpMissDays,BpMissInterval,BpEstDays,BpIncCurrDays,BpIncPastDays;
      String BpAbIncPct,BpDecCurrDays,BpDecPastDays,BpAbDecPct;

      Water.DbBean db = new Water.DbBean();
      Water.MyCalendar calendar = new MyCalendar();

      String date = calendar.getDateMySQL();

      try
      {
```

```
            db.connect();
      }
      catch(Exception e)
      {
            String message = "Database Connection failure";
            message += e;
            System.err.println("Caught Exception: "+message);
      }

      rs = db.execResults(sql);
      rs.next();

      try
      {
            db.close();
      }
      catch(Exception e)
      {
            String message = "Database Connection failure";
            message += e;
            System.err.println("Caught Exception: "+message);
      }

      if (rs.first() == false)
      {
            out.println("There are no Functionalities Scheduled to
Run.");
            out.println("<BR>Please update the Report Configuration
page as necessary.");
      }

      else
      {

            BPMiss = rs.getString("BoolBpMiss");
            BPEst = rs.getString("BoolBpEst");
            BPInc = rs.getString("BoolBpAbInc");
            BPDec = rs.getString("BoolBpAbDec");
            UsgMtSz = rs.getString("BoolUsgMtSz");
            UsgAcct = rs.getString("BoolUsgAcctTyp");
            MtWatch = rs.getString("BoolMtWatch");
            EndDate = rs.getString("EndDate");
            BpMissDays = rs.getString("BpMissDays");
            BpMissInterval = rs.getString("BpMissInterval");
            BpEstDays = rs.getString("BpEstDays");
            BpIncCurrDays = rs.getString("BpIncCurrDays");
            BpIncPastDays = rs.getString("BpIncPastDays");
            BpAbIncPct = rs.getString("BpAbIncPct");
            BpDecCurrDays = rs.getString("BpDecCurrDays");
            BpDecPastDays = rs.getString("BpDecPastDays");
            BpAbDecPct = rs.getString("BpAbDecPct");


            if (BPMiss.equals("true"))
            {
                  Water.BPMiss bpMiss = new Water.BPMiss();
                  bpMiss.setParam();
```

```
                bpMiss.run();
            }

            if (UsgMtSz.equals("true") || UsgAcct.equals("true"))
            {
                Water.UsagePattern usage = new UsagePattern();
                usage.setParam();
                usage.run();
            }

            if ( BPMiss.equals("true") || BPEst.equals("true") ||
BPInc.equals("true") || BPDec.equals("true"))
            {
                 String sql1 = "UPDATE BPConfig SET SysDate =
'"+date+"',EndDate = '"+EndDate+"',BPMiss = '"+BPMiss+"',"+

      "BPEst = '"+BPEst+"',BPInc = '"+BPInc+"',"+

      "BPDec = '"+BPDec+"',"+
                                                "BpMissInterval =
'"+BpMissInterval+"',BpMissDays = '"+BpMissDays+"',"+
                                                "BpEstDays =
'"+BpEstDays+"',BpIncCurrDays = '"+BpIncCurrDays+"',"+
                                                "BpIncPastDays =
'"+BpIncPastDays+"',BpAbIncPct = '"+BpAbIncPct+"',"+
                                                "BpDecCurrDays =
'"+BpDecCurrDays+"',BpDecPastDays = '"+BpDecPastDays+"',"+
                                                "BpAbDecPct =
'"+BpAbDecPct+"'";



                Water.DbBean db1 = new Water.DbBean();

                try
                {
                      db1.connect();
                      db1.execUpdate(sql1);
                      db1.close();
                }
                catch(Exception e)
                {
                String message = "Database Connection failure";
                message += e;
                }
            }

            if (BPEst.equals("true"))
            {
                Water.BPEst bpEst = new Water.BPEst();
                bpEst.setParam();
                bpEst.run();
            }

            if (BPInc.equals("true"))
            {
                Water.BPIncrease bpIncrease = new Water.BPIncrease();
```

```
                    bpIncrease.setParam();
                    bpIncrease.run();
              }

              if (BPDec.equals("true"))
              {
                    Water.BPDecrease bpDecrease = new Water.BPDecrease();
                    bpDecrease.setParam();
                    bpDecrease.run();
              }

              if (MtWatch.equals("true"))
              {
                    Water.MeterWatch mtrWatch = new Water.MeterWatch();
                    mtrWatch.setParam();
                    mtrWatch.run();
              }

       }

%>

<jsp:forward page= "home.jsp" />

</body>
</html>
```

8.2.2 BPIncrease.java

This is the javabean that contains the business logic for the BPAI functionality and which does all the processing with the data.

```
package Water;

import java.sql.*;
import java.util.*;


public class BPIncrease
{
      private int avgPaUse;
      private int avgCuUse;
      private String acctNo;
      private String suffix;
      private String bpAbIncPct;
      private String bpIncCurrDays;
      private String bpIncPastDays;
      private String endDate;
      private DbBean db;
      private String BgColor = "#C0C0C0";

      public BPIncrease()
      {}
```

```
	public void setParam()
	{
		String sqlParam = "SELECT
EndDate,BpIncCurrDays,BpIncPastDays,BpAbIncPct FROM Config";
		ResultSet rs=null;

		DbBean db1 = new DbBean();

		try
		{
			db1.connect();
			rs = db1.execResults(sqlParam);
			rs.next();
		}
		catch(Exception e)
		{
			String message = "Database Connection failure";
			message += e;
			System.err.println("Caught Exception: "+message);
		}



		try
		{
			db1.close();

			bpIncCurrDays = rs.getString("BpIncCurrDays");
			bpIncPastDays = rs.getString("BpIncPastDays");
			bpAbIncPct = rs.getString("BpAbIncPct");
			endDate = rs.getString("EndDate");
		}
		catch(Exception e)
		{
			String message = "Database Connection failure";
			message += e;
			System.err.println("Caught Exception: "+message);
		}

	}


	public void run()
	{
		String sqlPastDates;
		String sqlPastMin;
		String sqlPastMax;
		String sqlPastAvg;

		String sqlCurrDates;
		String sqlCurrMin;
		String sqlCurrMax;
		String sqlCurrAvg;

		String sqlDropCurrDates;
```

```
            String sqlDropCurrMin;
            String sqlDropCurrMax;
            String sqlDropCurrAvg;

            String sqlDropPastDates;
            String sqlDropPastMin;
            String sqlDropPastMax;
            String sqlDropPastAvg;

            String sqlInc;

            String sqlAddCol;
            String sqlUpdateConfig;
            String sqlDropResult;

            sqlPastDates = "CREATE TABLE BPPastDates AS "+
                                "SELECT AcctNo,Suffix,Max(ReadDate)
max,Min(ReadDate) min "+
                                "FROM MeterRead "+
                                "WHERE to_days(DATE_SUB('"+endDate+"',
INTERVAL "+bpIncCurrDays+" DAY)) - to_days(ReadDate) <=
"+bpIncPastDays+" "+
                                "AND to_days(DATE_SUB('"+endDate+"',
INTERVAL "+bpIncCurrDays+" DAY)) - to_days(ReadDate) >= 0 "+
                                "AND AcctNo like'FAKE%' "+
                                "AND Reading > 0 "+
                                "GROUP BY AcctNo,Suffix";

            sqlPastMin = "CREATE TABLE BPPastMin AS "+
                                "SELECT m.AcctNo
AcctNo,m.Suffix,m.Reading minRead,d.min minDate "+
                                "FROM MeterRead m,BPPastDates d "+
                                "WHERE d.min = m.ReadDate";

            sqlPastMax = "CREATE TABLE BPPastMax AS "+
                                "SELECT m.AcctNo
AcctNo,m.Suffix,m.Reading maxRead,d.max maxDate "+
                                "FROM MeterRead m,BPPastDates d "+
                                "WHERE d.max = m.ReadDate";

            sqlPastAvg = "CREATE TABLE BPPastAvg "+
                                "SELECT
min.AcctNo,min.Suffix,(max.maxRead - min.minRead)/(to_days(max.maxDate)
- to_days(min.minDate)) AvgPastUsage "+
                                "FROM BPPastMin min,BPPastMax max "+
                                "WHERE min.AcctNo = max.AcctNo "+
                                "AND min.Suffix = max.Suffix";

            sqlCurrDates = "CREATE TABLE BPCurrDates AS "+
                                "SELECT AcctNo,Suffix,Max(ReadDate)
max,Min(ReadDate) min "+
                                "FROM MeterRead "+
                                "WHERE to_days('"+endDate+"') -
to_days(ReadDate) <= "+bpIncCurrDays+" "+
                                "AND to_days('"+endDate+"') -
to_days(ReadDate) >= 0 "+
                                "AND AcctNo like'FAKE%' "+
```

```
                                    "AND Reading > 0 "+
                                    "GROUP BY AcctNo,Suffix";

            sqlCurrMin = "CREATE TABLE BPCurrMin AS "+
                                "SELECT m.AcctNo AcctNo,m.Suffix,m.Reading
minRead,d.min minDate "+
                                "FROM MeterRead m,BPCurrDates d "+
                                "WHERE d.min = m.ReadDate";

            sqlCurrMax = "CREATE TABLE BPCurrMax AS "+
                                  "SELECT m.AcctNo
AcctNo,m.Suffix,m.Reading maxRead,d.max maxDate "+
                                  "FROM MeterRead m,BPCurrDates d "+
                                  "WHERE d.max = m.ReadDate";

            sqlCurrAvg = "CREATE TABLE BPCurrAvg AS "+
                                "SELECT min.AcctNo,min.Suffix,(max.maxRead
- min.minRead)/(to_days(max.maxDate) - to_days(min.minDate))
AvgCurrUsage "+
                                "FROM BPCurrMin min,BPCurrMax max "+
                                "WHERE min.AcctNo = max.AcctNo "+
                                "AND min.Suffix = max.Suffix";

            sqlInc = "CREATE TABLE BPIncResults AS "+
                              "SELECT p.AcctNo
Account,p.Suffix,p.AvgPastUsage,c.AvgCurrUsage,((c.AvgCurrUsage -
p.AvgPastUsage)/p.AvgPastUsage * 100) PercentInc "+
                              "FROM BPPastAvg p,BPCurrAvg c "+
                              "WHERE p.AcctNo = c.AcctNo "+
                              "AND p.Suffix = c.Suffix "+
                              "HAVING PercentInc > '"+bpAbIncPct+"' ";



            sqlDropCurrDates = "DROP TABLE BPCurrDates ";
            sqlDropCurrMin = "DROP TABLE BPCurrMin ";
            sqlDropCurrMax = "DROP TABLE BPCurrMax ";
            sqlDropCurrAvg = "DROP TABLE BPCurrAvg ";

            sqlDropPastDates = "DROP TABLE BPPastDates ";
            sqlDropPastMin = "DROP TABLE BPPastMin ";
            sqlDropPastMax = "DROP TABLE BPPastMax ";
            sqlDropPastAvg = "DROP TABLE BPPastAvg ";

            sqlDropResult = "DROP TABLE BPIncResults";

            sqlAddCol = "ALTER TABLE BPIncResults ADD SysDate
datetime,ADD EndDate datetime ,ADD ConfigPercent int,ADD CurrDays
int,ADD PastDays int";

            sqlUpdateConfig = "UPDATE BPIncResults SET SysDate = '"+
new java.util.Date()+"',"+

"EndDate = '"+endDate+"',"+

"ConfigPercent = '"+bpAbIncPct+"',"+
```

```
"CurrDays = '"+bpIncCurrDays+"',"+

"Pastdays = '"+bpIncPastDays+"'";



            db = new DbBean();

            DBResults dbResult;

            try
            {
                  db.connect();
                  db.execUpdate(sqlDropResult);

                  db.execUpdate(sqlPastDates);
                  db.execUpdate(sqlPastMin);
                  db.execUpdate(sqlPastMax);
                  db.execUpdate(sqlPastAvg);

                  db.execUpdate(sqlCurrDates);
                  db.execUpdate(sqlCurrMin);
                  db.execUpdate(sqlCurrMax);
                  db.execUpdate(sqlCurrAvg);

                  db.execUpdate(sqlInc);

                  db.execUpdate(sqlDropCurrDates);
                  db.execUpdate(sqlDropCurrMin);
                  db.execUpdate(sqlDropCurrMax);
                  db.execUpdate(sqlDropCurrAvg);

                  db.execUpdate(sqlDropPastDates);
                  db.execUpdate(sqlDropPastMin);
                  db.execUpdate(sqlDropPastMax);
                  db.execUpdate(sqlDropPastAvg);

                  db.execUpdate(sqlAddCol);
                  db.execUpdate(sqlUpdateConfig);
                  db.close();
            }

            catch(Exception e)
            {
                  String message = "Database Connection failure";
                  message += e;
            }


      }
}
```

8.2.3 BillPrepResults.jsp

This is the JSP page, which is called when you wish to view the Bill Preparation results.

```
<html>

<head>
<title>Summary of Results</title>

</head>
<link rel="stylesheet" type="text/css"
href="../arlington_style_sheet.css">

<body>

<%@ page language="java" import= "java.sql.*, java.lang.*, Water.*" %>

<%    ResultSet rs;
      String sql = "SELECT EndDate,SysDate,BPMiss,BPEst,BPInc,BPDec
FROM BPConfig";
      String BgColor = "#C0C0C0";
      String BPMiss, BPEst, BPInc, BPDec,EndDate,SysDate;

      Water.DbBean db = new Water.DbBean();

      try
      {
            db.connect();
      }
      catch(Exception e)
      {
            String message = "Database Connection failure";
            message += e;
            System.err.println("Caught Exception: "+message);
      }

      rs = db.execResults(sql);
      rs.next();

      try
      {
            db.close();
      }
      catch(Exception e)
      {
            String message = "Database Connection failure";
            message += e;
            System.err.println("Caught Exception: "+message);
      }

      BPMiss = rs.getString("BPMiss");
      BPEst = rs.getString("BPEst");
      BPInc = rs.getString("BPInc");
      BPDec = rs.getString("BPDec");
      EndDate = rs.getString("EndDate");
      SysDate = rs.getString("SysDate");
%>
```

```
<div align="center">
  <center>
  <table border="0" cellspacing="4" cellpadding="2">
    <tr>
      <td colspan="3">
        <p align="center"><img border="0" src="../banner_water.gif"
width="708" height="49"></td>
    </tr>
    <tr>
      <td bgcolor="#000080" align="center"><font color="#FFFFFF"
face="Verdana" size="2"><b>Pavement
        System</b></font></td>
      <td bgcolor="#000080" align="center"><font color="#FFFFFF"
face="Verdana" size="2"><b>Permits
        System</b></font></td>
      <td align="center" style="font-family: Verdana; font-size: 10pt;
font-weight: bold" bgcolor="#000080"><b><a href="../home.jsp"><font
face="Verdana" size="2" color="#FFFFFF">Arlington
        Home</font></a></b></td>
    </tr>
  </table>
  </center>
</div>
<div align="center">
  <table border="1" width="722" bgcolor="#FFFFFF">
    <tr>
      <td width="100%">
        <div align="center">
          <center>
          <table border="0" width="650">
            <tr>
              <td>
                <p align="center"><img border="0" src="../drop.gif"
width="57" height="71"></p>
              </td>
              <td>
                <p align="center"><font face="Verdana" color="#000080"
size="5"><b>Water
                Management System Results</b></font></td>
              <td>
                <p align="center"><img border="0" src="../drop.gif"
width="57" height="71"></p>
              </td>
            </tr>
          </table>
          </center>
        </div>
      </td>
    </tr>
  </table>
</div>
<div align="center">
  <center>
<table border="1" width="722" bgcolor="#800000">
  <tr>
    <td width="100%">
      <p align="center"><font color="#FFFFFF" face="Verdana"
```

```
size="2"><b>Bill
      Prep Results</b></font></td>
  </tr>
</table>
  </center>
</div>
<p align="center">Current Date:<%= SysDate %></p>
<p align="center">    Following Analysis is based on:
<%= EndDate %> </p>

<h3 align="center"> </h3>
<hr>
<% }

      if (BPInc.equals("true"))
{ %>
<h3 align="center">Bill Prep: Abnormal Increase</h3>

<%

      String sqlIncHeader = "SELECT
SysDate,EndDate,BpAbIncPct,BpIncCurrDays,BpIncPastDays from BPConfig";
      String sqlIncTable = "SELECT
Account,Suffix,AvgPastUsage,AvgCurrUsage,PercentInc FROM BPIncResults";
      String IncBgColor = "#C0C0C0";
      Water.DbBean dbInc = new Water.DbBean();
      Water.DBResults dbIncResult;
      String urlInc = "BpAbIncGraph.jsp";
      ResultSet rsInc;

      try
      {     dbInc.connect();

      }

      catch(Exception e)
      {
            String message = "Database Connection failure";
            message += e;
      }

      rsInc = dbInc.execResults(sqlIncHeader);
      rsInc.next();

      if(rsInc.first()== true)
      {

            String sysDateInc = rsInc.getString("SysDate");
            String endDateInc = rsInc.getString("EndDate");
            String configPercent = rsInc.getString("BpAbIncPct");
            String currDaysInc = rsInc.getString("BpIncCurrDays");
            String pastDaysInc = rsInc.getString("BpIncPastDays");



            String htmlIncHeader = "<div align=\"center\">  <center> "+
                                    "<table border=\"0\"
```

```
width=\"490\">"+
                                          " <tr> <td width=\"25\"></td><td
width=\"449\">Threshold Set for Percent Increase:"+ configPercent +" %
</td> </tr>"+
                                          " <tr> <td width=\"25\"></td>
<td width=\"449\">Past Time Period set: "+pastDaysInc+" days</td>
</tr>"+
                                        "<tr> <td width=\"25\"></td> <td
width=\"449\">Current Time Period set:"+currDaysInc+"  days</td></tr>"+
                                        " <tr> <td width=\"25\"></td><td
width=\"449\"></td> </tr>"+
                                        "  </table> ";


            dbIncResult = dbInc.getQueryResults(sqlIncTable,true);
            String htmlIncTable =
dbIncResult.toLinkHTMLTable(IncBgColor,urlInc);

            out.println(htmlIncHeader);

            if(dbIncResult.getRowCount() < 1)
            {
                  out.println("No Records Found ");
            }
            else
            {
                  out.println(htmlIncTable);
            }

      }

      else

      {
            out.println("No records Found ");
      }


}
%>


<h3 align="center"> </h3>
<hr>
<form method="GET" action="BillPrepICS.htm">
    <p align="center"><input type="submit" value="Create Monthly
Report" name="B1"></p>
</form>
<p align="center"> </p>
```