1.Report No. FHWA/OH-2004/006	2. Government Accession No.	3. Recipient's Catalog No.

4.Title and Subtitle Smart Sign Ordering System – Phase I	5. Report Date: May, 2004 6. Performing Organization Code
7. Author(s) Ping Yi, Yingcai Xiao	8. Performing Organization Report No.
9. Performing Organization Name and Address The University of Akron Department of Civil Engineering Akron, OH 44325-3905	10. Work Unit No. (TRAIS) 11. Contract or Grant No. State Job No. 14785(0)
12. Sponsoring Agency Name and Address Ohio Department of Transportation 1980 West Broad Street Columbus, OH 43223	13. Type of Report and Period Covered Final Report 14. Sponsoring Agency Code

15. Supplementary Notes
Prepared in Cooperation with the U.S. Department of Transportation, Federal Highway Administration

16. Abstract

The University of Akron has developed an on-line traffic sign ordering system, the Smart Sign Ordering System (SSOS), for the Ohio Department of Transportation (ODOT). The objective of SSOS is to increase the efficiency of the sign ordering process by (1) reducing labor costs due to extended review time, (2) organizing submitted orders on-line so that production schedule can be adjusted and material usage estimated, (3) enabling on-line cost estimation and (4) proving a means of data management for summary of orders and production.

The implemented SSOS is an efficient tool for data exchange between the districts, Central Office and the Sign Shop. It allows automated data entry during preparation of sign orders, and provides on-line data review and modification capabilities. In addition, it enables querying and sorting, and helps tracking the orders in the production and delivery phases. This tool improves work efficiency in the sign ordering by reducing human errors and speeding up the entire order-filling process.

17. Key Words	18. Distribution Statement
Traffic signs, Data exchange, Sign ordering, Sign order selection and tracking, ODOT districts, Central Office, Sign Shop	No Restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161

19. Security classif. (of this report)	20. Security Classif. (of this page)	21. No of Pages	22. Price
Unclassified	Unclassified		

Smart Sign Ordering System (Phase I)

State Job No. 14785(0)

FINAL REPORT

Prepared in Cooperation with the Ohio Department of Transportation and the U.S. Department of Transportation, Federal Highway Administration

May, 2004

by
Ping Yi, Yingcai Xiao, Natheer Khasawneh

The University of Akron

DISCLAIMER STATEMENT

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the Ohio Department of Transportation or the Federal Highway Administration. This report does not constitute a standard, specification or regulation.

Report No.	2. Contract or Grant No.	3. The Principal investigators numbers
FHWA/OH-2004/006A	State Job No. 14785(0)	Ping Yi, Yingcai Xiao

4.Title and Subtitle	5. Name of the Agency
Smart Sign Ordering System – Phase I	Department of Civil Engineering The University of Akron

6. EXECUTIVE SUMMARY

The Ohio Department of Transportation (ODOT) utilizes the Ohio Standard Sign Design Manual as a guidance of standards and specifications for the design and fabrication of traffic signs. Because of variations in the physical characteristics and applied graphics/text of the signs, this manual contains a large amount of complex information and uses sign codes, EMS numbers, and graphic charts to represent individual and collective features of the signs (such as size, color, legend, and material, etc). When ODOT districts make sign orders, engineers/technicians often need to look into the Manual before filling the information line-by-line in the order form. There is very little computer automation in the ordering process.

ODOT operates a Sign Shop to fabricate signs. The districts submit standard sign orders to the Sign Shop, and special sign orders to the Central Office for review before they are forwarded to the Sign Shop, where all approved orders are processed in production planning, fabrication, and subsequent shipping. Due to lack of data automation and efficient means of data exchange and management, errors often occur in the current ordering work process.

The University of Akron developed an on-line traffic-sign ordering system, the Smart Sign Ordering System (SSOS). The objective of the system is to help in the sign ordering process by (1) reducing labor costs due to extended review time, (2) organizing submitted orders on-line so that production schedule can be adjusted and material usage estimated, (3) enabling on-line cost estimation and (4) proving a means of data management for summary of orders and production.

SSOS is an efficient tool for data exchange between the districts, Central Office and the Sign Shop. It has automated functions for data entry during preparation of sign orders, and provides on-line data review and modification capabilities. In addition, it enables querying and sorting, and helps tracking the orders in the production and delivery phases. This system improves the work efficiency in the sign ordering process, by reducing human errors and thus speeding up the entire order-filling process.

As the name suggests, SSOS is designed primarily for sign ordering management. To make the system more useful, such as facilitating sign production management in the Sign Shop, further development is needed to expand its functional features. The project team is currently discussing with the Sign Shop about such needs.

The entire project produced two products, this project report and a companion software CD containing the SSOS software program, the setup scripts, the source code and documentations.

This project report describes the need, objectives, system architecture, database design, and functionality of the system. The software program is supported by an on-line help which is built into it for convenient access by the users. The SSOS system runs over the ODOT Intranet backbone with the support of the central database managed by ODOT's Office of Information Technologies (IT). Because of this, user account management and operation of the software is under close supervision of the IT Office.

The pilot testing of the SSOS program followed several on-site training sessions for ODOT personnel from its districts, Central Office, and the Sign Shop. Preliminary results of the testing have shown that use of SSOS for sign ordering is feasible, convenient, and efficient. Limitations with the program have also been identified, which can be eliminated in future work to make the tool more effective and user-friendly.

7. For Copies of this Report, Contact:

Ohio Department of Transportation, Ms. Monique Evans, Office of Research and Development, (614) 728 - 6048, mevans@odot.state.oh.us

ACKNOWLEDGEMENTS

We want to express our sincere appreciations to the SSOS project team, led by ODOT project liaison, Mr. Jim Roth, for their active participation throughout the project. Our deep gratitude goes to Mr. Paul Trapasso, for organizing and hosting project discussions and training sessions, and providing his full support to the project development effort. While representatives from each involved ODOT office have all contributed to the project, we want to especially recognize Mr. Michael Orndorf and Mr. Bill Puckett and their teams, for their work in the designing, development, testing, training, and deployment phases of the project.

Our special thanks go to Mr. Mohammad Khan, for his guidance and help to make this project possible.

Support to the project from the College of Arts and Sciences, the College of Engineering, and the Office of Research Programs at The University of Akron in student wages, tuitions, and lab facilities is acknowledged.

PREFACE

"Smart Sign Ordering System" (SSOS) is an on-line traffic-sign ordering system developed by The University of Akron for The Ohio Department of Transportation (ODOT). Internet and database technologies, including Java, JDBC, Servlet/JSP, HTML, and JavaScript were used to construct SSOS as a multi-tier J2EE-like application. This report presents the technical details of the system, including its system architecture, database design, business model, and code implementation.

TABLE OF CONTENTS

Chapter I

BACKGROUND

1.1 Sign Ordering and the Existing Problem

Traffic signs are a very important tool for traffic control and transportation system management. New traffic signs are needed in every ODOT district either because of new roadway constructions or old sign replacements. The districts submit standard sign orders to the Sign Shop, or to the Central Office for review of special signs before they are forwarded to the Sign Shop, where all approved orders are processed in production planning, fabrication, and subsequent shipping. Due to lack of data automation and efficient means of information exchange and management between the districts, Central Office, and the Sign Shop, order errors often affect timely production and delivery of signs.

As the local conditions in each district are different, manual input of data (codes numbers, EMS numbers, sizes, colors, etc.) for different signs is usually slow and requires very focused attention, and it often relies on memorization of the codes and numbers from the Standard Sign Design Manual. Despite careful efforts by the districts, inconsistencies are often found in the orders of special signs submitted to the Central Office. The reviewing work at Central Office must therefore be conducted thoroughly to ensure conformation of the design to the standards. In the production process of signs at the Sign Shop, the production schedule for a certain type of sign depends on the quantity ordered and available material. Generally, the Sign Shop is unaware of the collective demand for different types

of signs coming from the districts; likewise, when orders are initiated, the individual districts do not know the production schedule of the types of signs they are ordering. Delays often occur due to orders backlog or material shortage at the Sign Shop.

In order to reduce the chances for errors by the districts where orders are requested, and shorten the amount of time needed by the Central Office to review special orders, in November of 1998, a committee was formed involving ODOT districts, Central Office, and the Sign Shop, to discuss improvement over the existing method for ordering signs and data processing. The committee called for the development of a semi-automated sign ordering system, with which engineers and technicians at the districts and Central Office can order and review signs with data automation, whereas the Sign Shop will be able to review and group the orders on-line to improve production planning and material management.

1.2 Project Objectives

The overall goal of the SSOS program is to improve the efficiency of the sign ordering process by reducing errors in orders, speeding review time, and modifying order submissions to align better with the current sign production method used by the Sign Shop. To achieve this goal, the following objectives are proposed to meet ODOT's requirements:

1) To reduce labor cost due to extended review time to check data accuracy by building into SSOS design standards (for example, coding, sizing, and material use), and allowing modification of orders on-line over the network.

2) To organize submitted orders to meet the needs of the Sign Shop for production of the signs by providing ways to adjust production schedule and estimate material usage, and group the orders by types of signs, materials used, and fabrication methods, etc.

3) To enable on-line cost estimation by setting up an automated cost analysis routine which summarizes and updates the cost of each added or changed order.

4) To provide a means of data management so that summary of orders and sign productions can be easily generated by the users.

To help achieve the above objectives, SSOS is constructed as a Web-based and N-tier on-line software and utilizes the latest Internet and database technologies including Java, Object-Oriented Design, JDBC and Servlet for back end, JSP, custom tag library and JavaScript for front-end representation. These technologies provide the foundations for the development of SSOS in information utilization locally and data flow between the districts, Central Office, and the Sign Shop, thus to ensure design uniformity and consistency of traffic signs.

Chapter II

SYSTEM SPECIFICATIONS

2.1 Functional Specifications

2.1.1 Order Lifecycle

An order is initially created on-line at a district office and sent to the Central Office for approval or directly to the Sign Shop without approval if it is a bypass-sign type like warehouse signs. The Sign Shop will handle the manufacturing based on material and other resources, and packaging based on district. District will conclude the order when the signs are received and archive it to history. During the lifecycle of the order, three units can communicate about the order with each other by leaving comments in the order history field. Every status change to the order will also be automatically documented in the history field. For the purpose of reducing order latency, the project committee has determined that only one order item will be permitted in each order.

2.1.2 District Office

A District Office unit has three functions. First of all, it will gather the need of signs from either counties belonging to the district or its local warehouse, then place an order on-line. Secondly, it will track the order status while the order goes through the approval and fabrication process. Thirdly, upon the receipt of the manufactured signs, it will have the responsibility to confirm the reception and archive the orders through the on-line system.

When placing a new order, the operator will have three ways to locate a pre-specified sign design in the system: The technical specifications of the signs are imported from the JIMANI program at the Sign Shop. These specifications can be accessed on-line by its EMS number, by its sign code, or by the legend on the sign. The user will navigate with the three choices to come to an ordering page. SSOS will automatically fill the order form with all sign-specific features–whichever applies, including background color, applied color, size, legend, and material, etc.. The user then specifies the quantity and other information related to the order. The total cost of the order is automatically computed based on the quantity ordered and the unit price of the sign. The unit price information is pre-determined by the Sign Shop, and can be changed on the order form if necessary.. The total cost is automatically adjusted when the user modifies the quantity ordered.

After the order is initially created it will be in a pending status for further modification by the district operator until it is finally submitted. Viewing of a previously submitted order is constrained to showing the order content with no modification permission to the district operator. When the order status changed to “delivered” and the district manager does receive the order, a confirm reception button will shown and he can choose to archive this order after confirmation.

2.1.3 Central Office

The main function of the Central Office is to validate orders which are open for designation, such as fully designable signs and legend designable signs. The bypass signs

such as warehouse sings whose attributes are all fixed will bypass the inspection of the Central Office and will be directly sent to the Sign Shop for manufacture.

Central Office can change the condition for the bypass and specify which signs can bypass the Central Office approval and which signs should be reviewed by the Central Office. Central Office will have the function to review all orders whose status is “submitted” for inspection and have an approve button thereafter. Central Office will also have the permission to modify disqualified orders.

For the purpose of material and resource management, the Central Office needs a generic search function that can return the certain group of orders by the combination of district, material type, manufacture method, etc. This function is very useful when searching documented orders or creating summaries upon various attributes such as certain district or within certain dates.

2.1.4 Sign Shop

When the approved orders arrive at the Sign Shop, they need to group the orders by types of signs, materials used, and fabrication methods, etc., to facilitate production planning and material usage estimation. The Sign Shop has three sub-groups: production planning group, packaging group and transferring group.

2.1.4.1 Planning Group

Planning group will handle receiving orders and make a production plan of them based on material and production method. It will also confirm the completion of the production process.

2.1.4.2 Packaging Group

Packaging group will gather all finished signs from manufacturing/warehouse and package them for the delivering to the districts.

2.1.4.3 Transferring Group

Transferring group will enter the delivered orders in the old EMS system for the record keeping purpose.

All of the three sub-groups will have the view-order function and the generic searching function. Those are critical to all of them because planning will need them to group the orders cross districts based on the production methods and packaging will need them to regroup the orders by districts for delivering.

2.1.5 Other Functions

2.1.5.1 Order Summary

Basic order summary capabilities are provided to District, Central Office and Sign Shop modules to summarize ordering statistics of total number of signs ordered and total price of the orders. The summary can be generated by grouping orders based on order status

(submitted, approved, completed, delivered, etc.), type of material, or production method used. Comprehensive summaries can be generated through customizable queries using any query conditions. Since SSOS uses ODOT's Sybase database system which can be accessed through GQL (Graphical Query Language), other summaries or reports can be generated using GQL without affecting the internal coding of the system.

2.1.5.2 User Accounts

Adding users to the system and managing the access control are done through the Onelogin security system in use at ODOT. File security system types were used. To add a user to the system the Onelogin user name should be added along with a string that lists the user's access to the system and the groups the user belongs to. The string contains one or more groups delimited with space. The first group in the string is considered the default group where the user is assigned to once he/she logs into the system and he/she can switch between groups from the toolbars in the top of his/her screen.

The name of the groups are: "district" for creating orders, "central" for approving orders, "planning" for planning for productions and completing productions, "packing" for delivering orders, and "transfer" for archiving orders into the EMS database.

Chapter III

System Design

3.1 Java Technology

The Java programming language is a high-level language that can be characterized by the features listed in Table 3.1. The Java programming language is different from many others in that the program is both compiled and interpreted. With the compiler, a program is first translated into an intermediate language called *Java Bytecode* —the platform-independent code interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed.

• Simple	• Architecture Neutral
• Object Oriented	• Portable
• Distributed	• High Performance
• Interpreted	• Multithreaded
• Robust	• Dynamic
• Secure	

Table 3.1 Advantages of Java Language

You can think of Java bytecode as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it is a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java bytecode helps make "write once, run anywhere" possible—as long as a computer has a Java VM, the same

program written in the Java programming language can run on a Windows based PC, a Solaris workstation, or on an iMac [1].

A *platform* is the hardware or software environment in which a program runs. Some of the most popular platforms are Windows, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components: The *Java Virtual Machine* (Java VM) and the *Java Application Programming Interface* (Java API). The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. Figure 3.1 depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware [1].

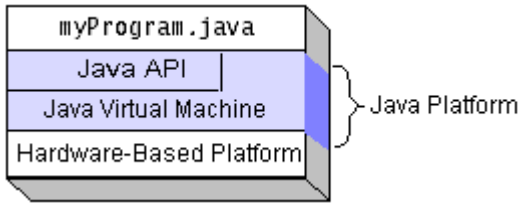

Figure 3.1 Java Runtime Layers

3.2 Servlet/JSP Technology

A servlet can be thought of as a server-side applet at a superficial level. Servlets are loaded and executed by a web server in the same manner that applets are loaded and executed by a web browser.

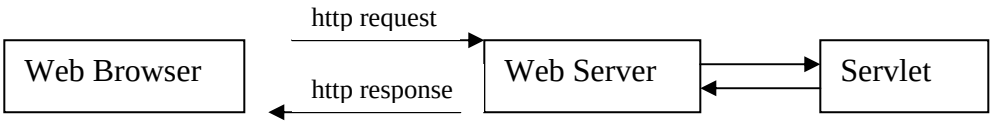

Figure 3.2 Basic Servlet Flow

As shown in Figure 3.2, the client (most likely a web browser) makes a request via HTTP, the web server receives the request and forwards it to the servlet. If the servlet has not yet been loaded, the web server will load it into the Java Virtual Machine and execute it. The servlet will receive the HTTP request and perform some type of process. The servlet will return a response back to the web server. The web server will forward the response to the client. Compared to most of the server side languages, servlets have the following advantages [2]:

1. Servlets are persistent. Servlets are loaded only once by the web server and can maintain services (such as a database connection) between requests. CGI scripts, on the other hand, are transient. Each time a request is made to a CGI script, it must be loaded and executed by the web server. When the CGI script is complete, it is removed from memory and the results are returned to the client. All program initialization (such as connecting to a database) must be repeated each time a CGI script is used.

2. Servlets are fast. Since servlets only need to be loaded once, they offer much better performance over their CGI counterparts.

3. Servlets are platform independent. As mentioned before, servlets are written in java, which inherently brings platform independence to your development effort.

4. Servlets are extensible. Since servlets are written in java, this brings all of the other benefits of java to your servlet. Java is a robust, object-oriented programming language, which easily can be extended to suit your needs.

5. Servlets are secure. The web browser does not communicate directly with a servlet. The servlet is loaded and executed by the web server. This brings a high level of security. This means that if the web server is secure behind a firewall, then your servlet is secure as well.

3.3 J2EE Technology

J2EE multi-tiered applications are generally considered to be three-tiered applications because they are distributed over three different locations: client machines, the J2EE server machine, and the database or legacy machines at the back end. Three-tiered applications that run in this way extend the standard two-tiered client and server model by placing a multithreaded application server between the client application and back-end storage.

J2EE applications are made up of components. A *J2EE component* is a self-contained functional software unit that is assembled into a J2EE application with its related

classes and files and that communicates with other components. The J2EE specification defines the following J2EE components:

- Application clients and applets are components that run on the client.
- Java Servlet and JavaServer Pages technology components are Web components that run on the server.
- Enterprise JavaBeans components (enterprise beans) are business components that run on the server.

J2EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server. A J2EE client can be a Web client or an application client.

Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before a Web, enterprise bean, or application client component can be executed, it must be assembled into a J2EE application and deployed into its container. Enterprise JavaBeans (EJB) container manages the execution of enterprise beans for J2EE applications and Web container manages the execution of JSP page and servlet components for J2EE applications. The utilization of

containers on J2EE server gives the J2EE applications sturdy, managed, and secure transactions [3].

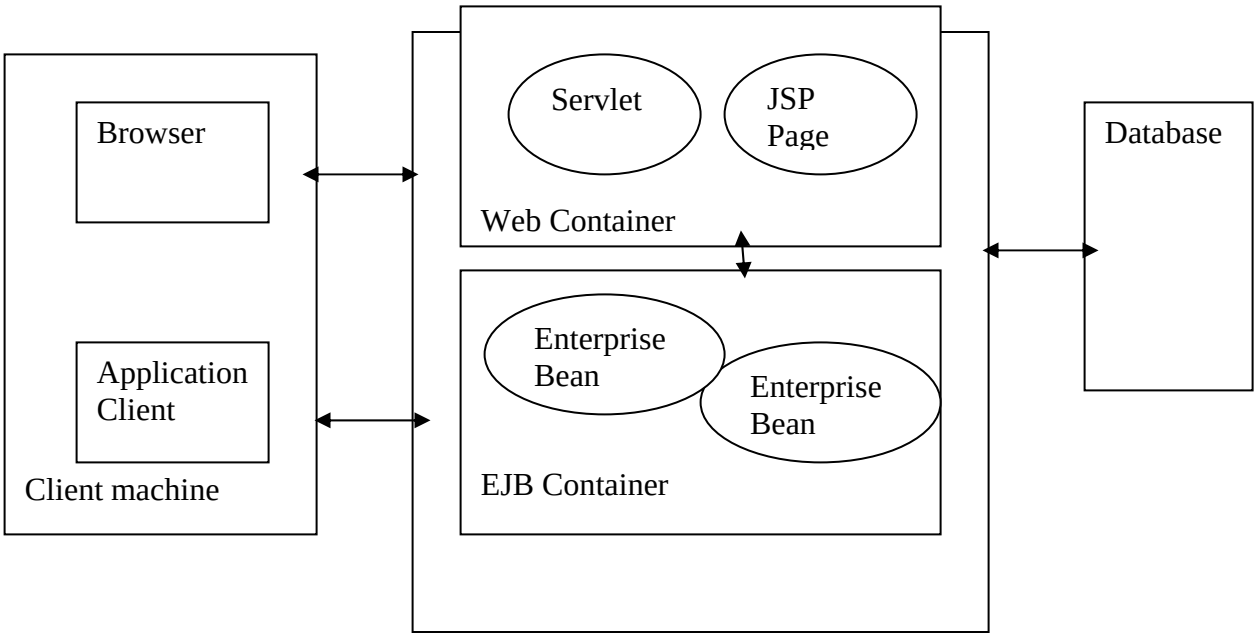

Figure 3.3 J2EE Architecture

3.4 HTML/Java Script

HTML stands for the Hypertext Markup Language. It is not an Internet protocol. In practical terms, HTML is a collection of platform-independent styles (indicated by markup tags) that define the various components of a World Wide Web document. HTML is the major language of the Internet's World Wide Web. Web sites and web pages are written in HTML. HTML consists of a set of tags and internal commands that are embedded inside Web pages to control the appearance and layout of Web pages, as well as links to other Web pages.

JavaScript, as the name suggests, is a scripting language, which is being developed by Netscape. JavaScript is object-based scripting language for client and server applications. JavaScript lets you create applications that run over the Internet. It is interpreted, weakly typed, over-permissive, embedded into an application, and serves as glue to hold together components written in other languages. Client applications run in a browser, such as Netscape Navigator, and server applications run on a server. Using JavaScript, you can create dynamic HTML pages that process user input and maintain persistent data using special objects, files, and relational databases.

With JavaScript you can easily create interactive web pages. Since the JavaScript is not totally compatible between Netscape and Microsoft Internet Explorer, here we implemented it in Microsoft Internet Explorer 5.0 or later.

You can embed JavaScript in an HTML document in the following ways:

1. As statements and functions within a <SCRIPT> tag.

2. By specifying a file as the JavaScript source (rather than embedding the JavaScript in the HTML).

3. By specifying a JavaScript expression as the value of an HTML attribute as event handlers within certain other HTML tags (mostly form elements).

Unlike HTML, JavaScript is case sensitive. JavaScript can be thought of as an extension to HTML that allows authors to incorporate some functionality in their web pages.

JavaScript and Java are similar in some ways but fundamentally different in others. The JavaScript language resembles Java but does not have Java's static typing and strong type checking. JavaScript supports most Java expression syntax and basic control-flow constructs. In contrast to Java's compile-time system of classes built by declarations, JavaScript supports a runtime system based on a small number of data types representing numeric, Boolean, and string values. JavaScript has a prototype-based object model instead of the more common class-based object model. The prototype-based model provides dynamic inheritance; that is, what is inherited can vary for individual objects. JavaScript also supports functions without any special declarative requirements. Functions can be properties of objects, executing as loosely typed methods.

3.5 Tomcat JSP Server and Testing Environment

Tomcat is the servlet container used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies. For the nature of JSP, Tomcat has a JSP engine that will convert a JSP script to a servlet at run time and that the servlet is compiled and loaded into java virtual machine by the servlet engine in Tomcat.

Tomcat is an open source project developed by Jakarta group, which is a non-profit branch of Sun Microsystems. The Tomcat server we use is version 4.1. The server administrative tool is quite simple and practical: It is stored in XML descriptor files. Tomcat does not provide user interface for administrative management of the server, one has to

manually edit the XML file to have the server configured. (Setting user account and level, password, maximum thread number, etc).

3.6 Visual Age/WebSphere Debugging/Deploying Environment

While Jakarta Project members worked on the Tomcat project which is an open source API, IBM has reached a co-developing agreement with SUN for commercializing the Tomcat server. The Visual Age/Web sphere bundle deployed the core of Tomcat for the servlet/JSP server part, and builds many business feature and rich user interface as well as robust performance on it.

Visual Age is an IDE for Java programming that will meet all purposes: GUI, Multi-threaded, Servlet, Applet, JSP, Enterprise JavaBean, etc.. It provides excellent project management for team work and version control as well as good compile-on-the-fly and common debugging features. For web development, it provides a WebSphere testing environment that will do the debugging of a JSP script by compiling to servlet class and tracking the value and recording each line of response it generated. This is really a breakthrough because JSP is notoriously hard to debug like all the server side scripting languages. With the help of Visual Age, a programmer can debug a JSP page as a normal Java file. Generally speaking, with a persistence framework and unit test environment for WebSphere, VisualAge for Java provides a fast way to develop, test, and deploy end-to-end e-business applications.

WebSphere server is an integrated multi-platform application server that supports HTTP protocol, CGI and JSP scripting language. Overall, WebSphere application server provides stable transactions, secure data storage and XML support.

3.7 State Diagrams

A state diagram shows the sequences of states that an object or an interaction goes through during its life time [4]. Figure 3.4 shows the flow of an order in SSOS.

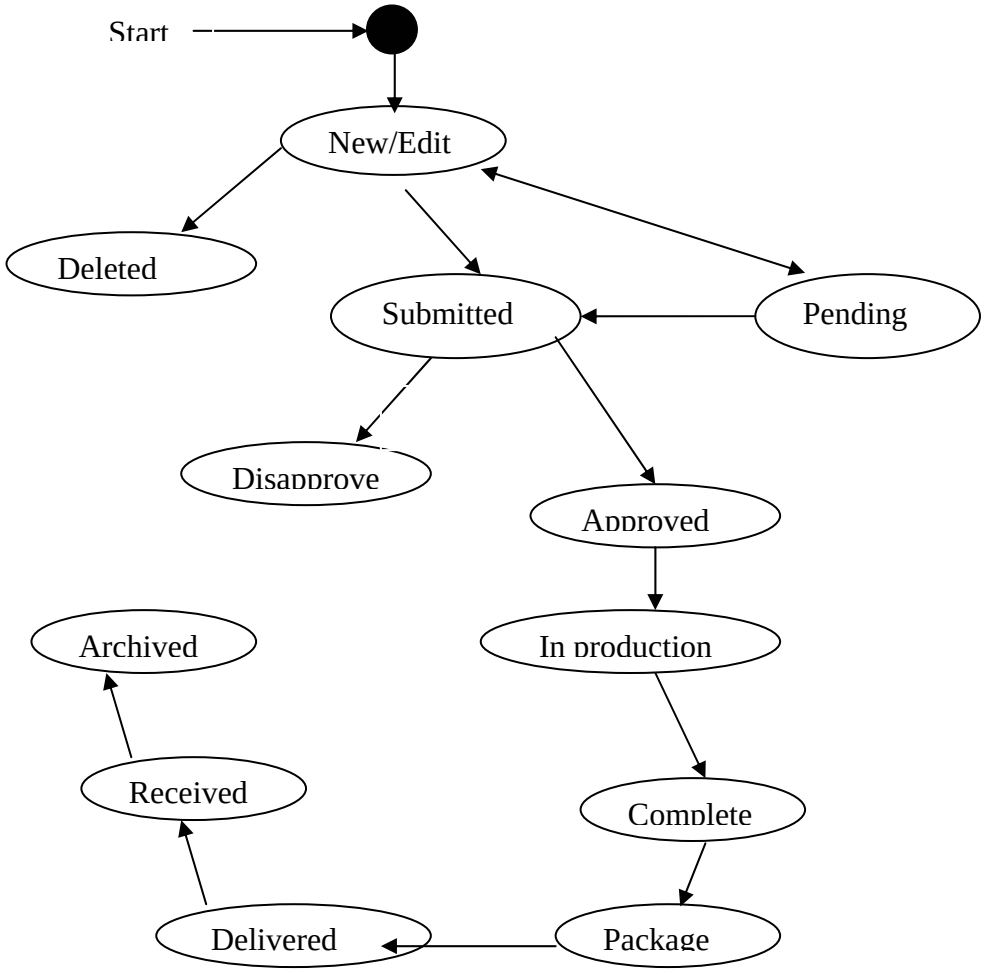

Figure 3.4 Order Processing State Diagram

A district office can start and end an order. The district office can modify a pending order but cannot do so after submission. Central Office can modify an order when it has not been approved, but cannot do so after the approval. Sign Shop can modify an order when the order comes in for fabrication but cannot do so when the order enters into production.

3.8 System Architecture

During the designing phase, J2EE architecture was exploited so as described in [3]. SSOS follows the architecture of J2EE, but does not require a J2EE container to run. SSOS can function in any application server container that supports Servlets/JSP. Tomcat is selected since it is supported by ODOT's DoIT.

We have several entity beans representing each database table and session beans for handling the order processing functions. We did not use java beans for information transferring between JSP and middleware. Instead, we used setter and getter functions in each bean class and object oriented parameter transferring of functions in a session beans classes. We found this design makes designing work quite schematic and implementing work very convenient. Since we do not use a real J2EE server, the beans are not in container, so the entity beans could not really access the database, thus we implemented a class called DBObjectManager which functions as a layer between entity beans and the database server. The whole schema is shown in Figure 3.5.

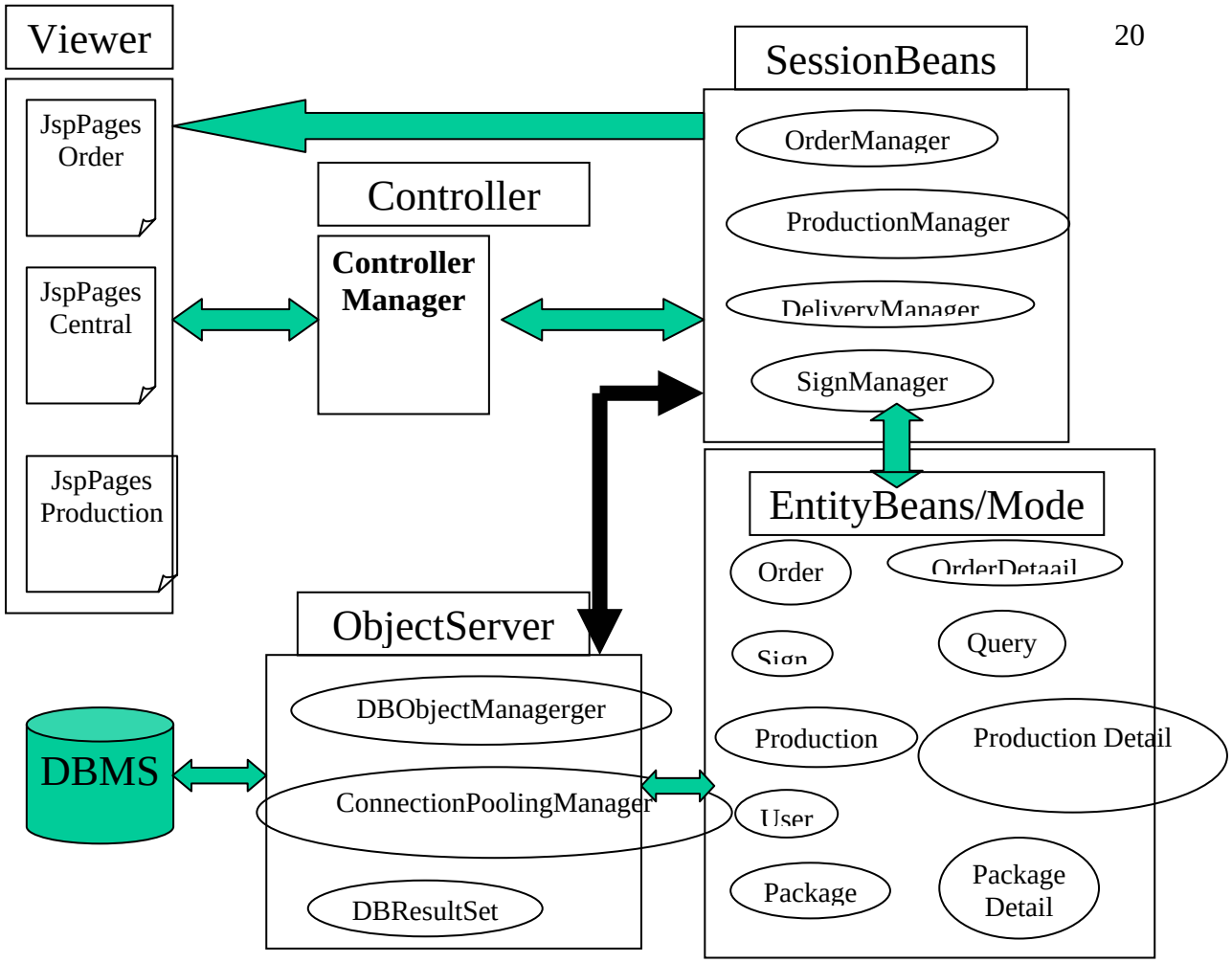

Figure 3.5 System Architecture

3.9 Use Case Diagram

A Use Case Diagram

- Describes the behavior of a system from a user's standpoint
- Has a functional description of a system and its major processes
- Provides a graphic description of who will use a system and what kinds of interactions to expect within that system

- Is a group of processes that occur within the application area
- Has actors which are entities outside the area that are going to use the application

The use of the Use Case Diagram shows the relationship among actors and use cases within a system [4]. The Use Case Diagram we will be creating tracks various functions and those who interact with the functions within the SSOS system.

Actors in SSOS can be divided into four groups:

- District, this group includes manager and operators in a district office;
- Central Office, users in this group can view orders from all the districts and validate them.
- Sign Shop, users in this group will check incoming orders, make plans for production, and ship completed orders to destinations;

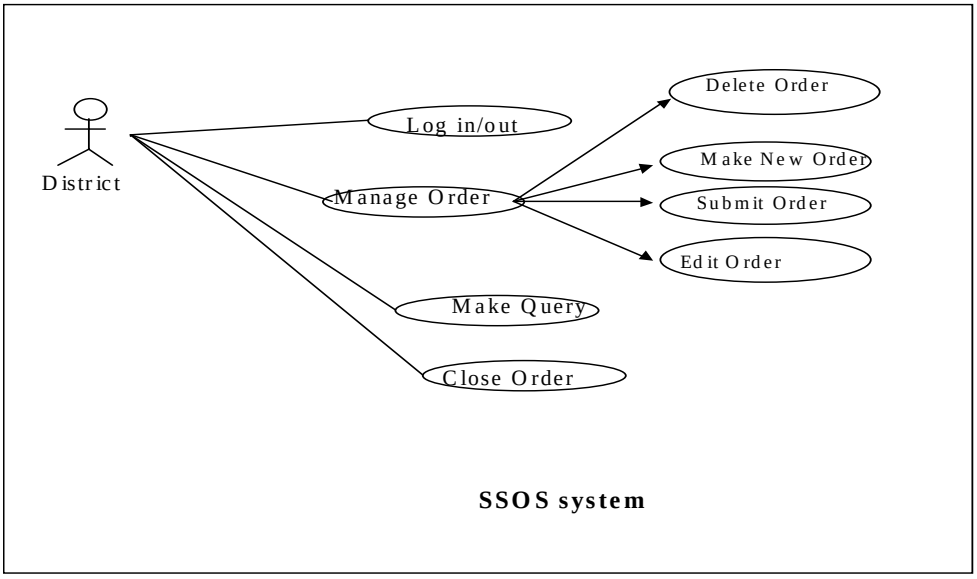

Figure 3.6 Use Case: District

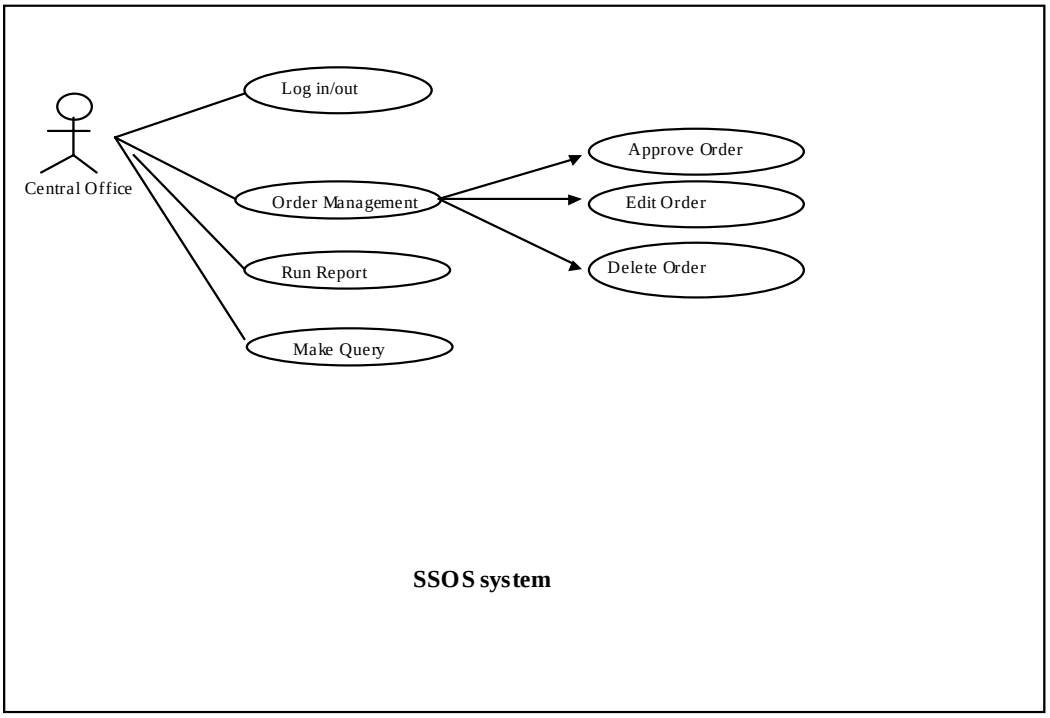

Figure 3.7 Use Case: Central Office

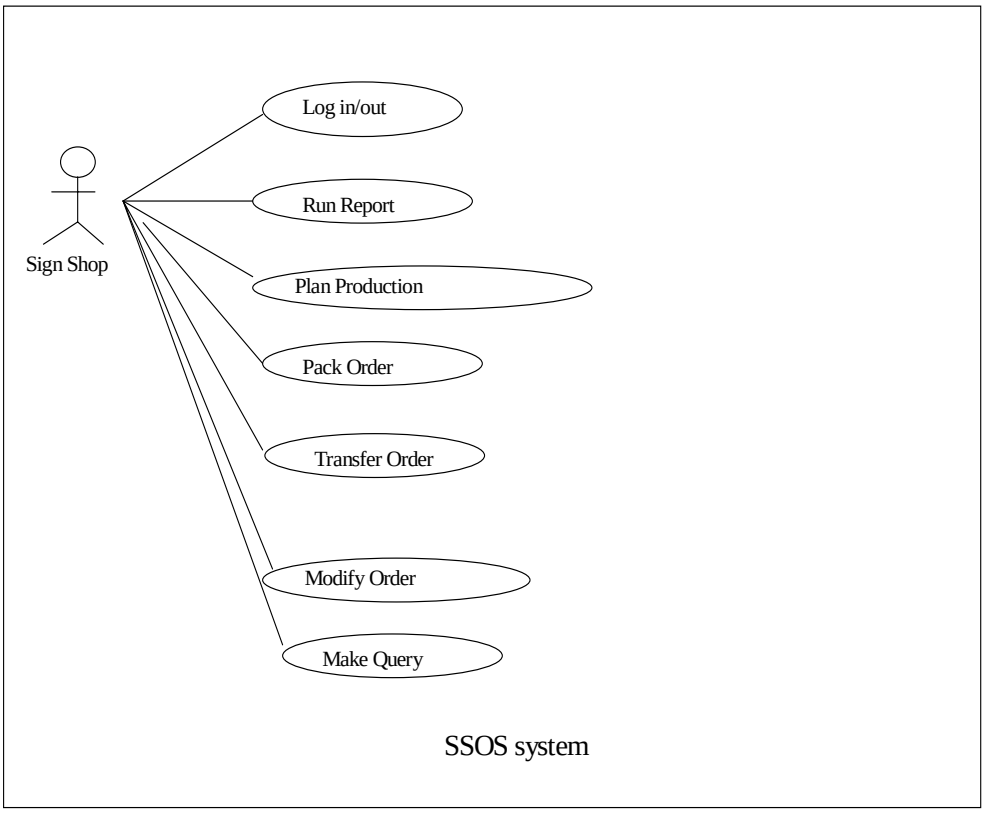

Figure 3.8 Use Case: Sign Shop

3.10 Database Schema

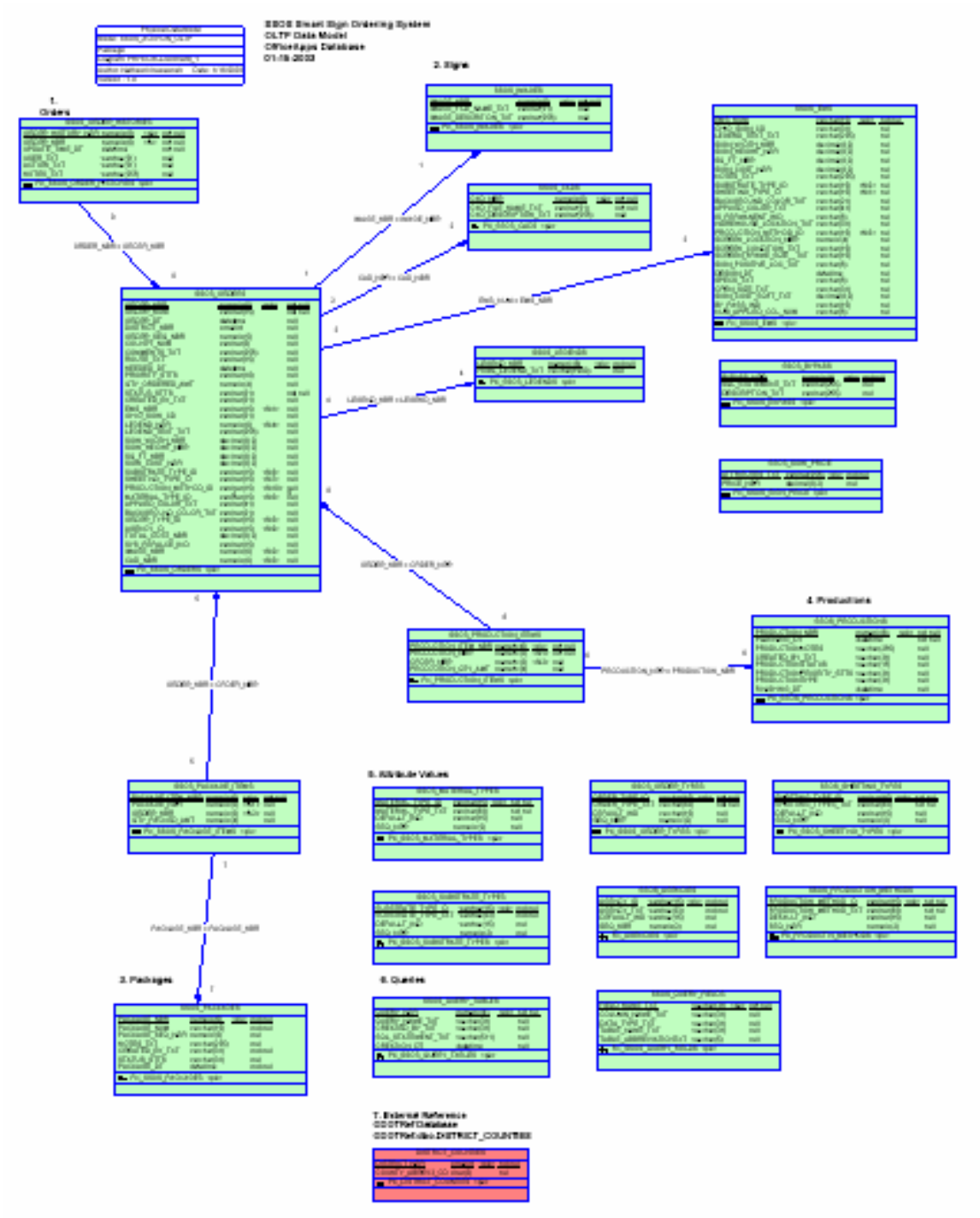

Figure 3.9 Database Schemas

(Note: full size image on the SSOS CD: ssos_reports\SSOSPhysicalDiagram.pdf)

Chapter IV

IMPLEMENTATION AND CODE EXPLANATION

4.1 Database Implementation

The database is implemented on Sybase database system, version 12.5. This DB server supports JDBC IV connection and standard SQL query. The table property and field names are listed below and they can be found in the file SSOSTableDefinition.pdf on the SSOS CD under ssos_reports directory. Scripts for setting up SSOS database tables can also be found in the same directory.

1. CreateTables.sql script will create all the necessary tables for SSOS.
2. looupdate.sql script will insert the necessary look up values.
3. ems.sql script will insert the sign information which has been exported from Jimani. (This script will take longer to run than the others)

ORDERS

SSOS_ORDER_HISTORIES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Order History Number*	SSOS_ORDER_HISTORIES / ORDER_HISTORY_NBR	numeric(8)	Unique number assigned to each record in the History table
Order Number	SSOS_ORDER_HISTORIES / ORDER_NBR	numeric(6)	Reference number to the SSOS_ORDERS table that refer to the ORDER that this history record belong to
Update Time Date	SSOS_ORDER_HISTORIES / UPDATE_TIME_DT	datetime	Time and date on which the History record added
User	SSOS_ORDER_HISTORIES / USER_TXT	varchar(31)	User name of the user who took the action for this history record
Action	SSOS_ORDER_HISTORIES / ACTION_TXT	varchar(31)	Action that took place for this history record
Notes	SSOS_ORDER_HISTORIES / NOTES_TXT	varchar(255)	Extra notes to elaborate on the action that took place when this record added

SSOS_ORDERS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Order Number*	SSOS_ORDERS / ORDER_NBR	numeric(6)	Unique number assigned to each order in SSOS
Order Number	SSOS_ORDERS / ORDER_NUM	varchar(15)	Unique number assigned to each order in SSOS. This number consists of 2-digit district numbers + '-' + 5 digit sequence number per district per fiscal year + '-' + 2- digit fiscal year. E.g. 1-00049-02 is order no. 49 in year 2002 for district 1
Order Number	SSOS_ORDERS / ORDER_DT	datetime	Time and date on the order have created
District Number	SSOS_ORDERS / DISTRICT_NBR	smallint	District number who created the order
Order Sequence Number	SSOS_ORDERS / ORDER_SEQ_NBR	numeric(5)	Order sequence per district per fiscal year
County Number	SSOS_ORDERS / COUNTY_NUM	varchar(3)	County number for which the order belong
Comments	SSOS_ORDERS / COMMENTS_TXT	varchar(255)	Comments on the order
Rout	SSOS_ORDERS / ROUTE_TXT	varchar(15)	Rout number where the order belong

Needed Date	SSOS_ORDERS / NEEDED_DT	Datetime	Date by which the order is needed
Priority	SSOS_ORDERS / PRIORITY_STTS	varchar(10)	Priority by the order is needed
Quantity Ordered	SSOS_ORDERS / QTY_ORDERED_AMT	numeric(4)	Number of sign that is needed by that order
Status	SSOS_ORDERS / STATUS_STTS	varchar(31)	Current status of the order
Created By	SSOS_ORDERS / CREATED_BY_TXT	varchar(31)	User name who created the order
EMS Number	SSOS_ORDERS / EMS_NBR	varchar(15)	EMS number of the sign that is ordered
Ohio Sign Code	SSOS_ORDERS / OHIO_SIGN_CD	varchar(31)	Ohio Sign code of the sign that is ordered
Federal Sign Code	SSOS_ORDERS / FED_SIGN_CD	varchar(31)	Federal Sing code of the sign that is ordered
Legend Number	SSOS_ORDERS / LEGEND_NBR	numeric(6)	Reference to the legend table
Legend Text	SSOS_ORDERS / LEGEND_TEXT_TXT	varchar(255)	Text only (no image) description of the legend
Sign Width	SSOS_ORDERS / SIGN_WIDTH_NBR	decimal(8,2)	Sign Width
Sing Height	SSOS_ORDERS / SIGN_HEIGHT_NBR	decimal(8,2)	Sign Height
Sign Foot	SSOS_ORDERS / SQ_FT_NBR	decimal(8,2)	Sign Size in Square Foot
Sign Cost	SSOS_ORDERS / SIGN_COST_NBR	decimal(8,2)	Sign Cost
Substrate	SSOS_ORDERS / SUBSTRATE_TXT	varchar(6)	Substrate type of the sign
Sheet	SSOS_ORDERS / SHEET_TXT	varchar(15)	Sheeting type of the sign
Background Color	SSOS_ORDERS / BACKGROUND_COLOR_TXT	varchar(21)	Sign Background color
Applied Color	SSOS_ORDERS / APPLIED_COLOR_TXT	varchar(41)	Sign Applied color
Production Method	SSOS_ORDERS / PRODUCTION_METHOD_TXT	varchar(15)	Sign Production Method
Material Type	SSOS_ORDERS / MATERIAL_TYPE_TXT	varchar(15)	Sign Material Type
Order Type	SSOS_ORDERS / ORDER_TYPE_TXT	varchar(15)	Type of the order like Standard, Special, Warehouse, etc.
			Name of the agency that is associated with order

Agency	SSOS_ORDERS / AGENCY_TXT	varchar(15)	
Total Cost	SSOS_ORDERS / TOTAL_COST_NBR	decimal(8,2)	Total cost of the order usually it is Individual Cost X Number of Signs
Systematic Replacement	SSOS_ORDERS / SYS_REPALCE_IND	varchar(15)	Determine whether the order is systematic replacement or not
Image Number	SSOS_ORDERS / IMAGE_NBR	numeric(6)	Reference to the image table
CAD Number	SSOS_ORDERS / CAD_NBR	numeric(6)	Reference to the Cad table

SIGNS

SSOS_IMAGES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Image Number*	SSOS_IMAGES / IMAGE_NBR	numeric(6)	Unique number assigned to each image
Image File Name	SSOS_IMAGES / IMAGE_FILE_NAME_TXT	varchar(31)	File name of the image, not including the path name
Image Description	SSOS_IMAGES / IMAGE_DESCRITION_TXT	varchar(255)	Description of the image

SSOS_CADS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
CAD Number*	SSOS_CADS / CAD_NBR	numeric(6)	Unique number assigned to each CAD
CAD File Name	SSOS_CADS / CAD_FILE_NAME_TXT	varchar(31)	File name of the cad, not including the path name
CAD Description	SSOS_CADS / CAD_DESCRIPTION_TXT	varchar(255)	Description of the CAD

SSOS_LEGENDS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Legend Number*	SSOS_LEGENDS / LEGEND_NBR	numeric(6)	Unique number assigned to each legend
HTML Legend	SSOS_LEGENDS / HTML_LEGEND_TXT	varchar(1900)	Legend description including HTML tags and code

SSOS_BYPASS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Bypass Number*	SSOS_BYPASS / BYPASS_NBR	numeric(4)	Unique number assigned to each Bypass definition
SQL Statement	SSOS_BYPASS / SQL_STATEMENT_TXT	varchar(255)	SQL Statement which select the signs which bypass the district
Description	SSOS_BYPASS / DESCRIPTION_TXT	varchar(255)	Description of the bypass definition

SSOS_SIGN_PRICE

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Attributes Text*	SSOS_SIGN_PRICE / ATTRIBUTES_TXT	varchar(255)	Attributes that select a group of signs based on Material Type, Production Method, Background Color, and Applied Color
Price	SSOS_SIGN_PRICE / PRICE_NBR	decimal(8,2)	Price of the group signs matched the attributed in previous field

SSOS_EMS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
EMS Number*	SSOS_EMS / EMS_NUM	varchar(15)	EMS number of the sign
Ohio Sign Code	SSOS_EMS / OHIO_SIGN_CD	varchar(31)	Ohio Sign Code of the sign
Federal Sign Code	SSOS_EMS / FED_SIGN_CD	varchar(31)	Federal Sign Code of sign
Legend Text	SSOS_EMS / LEGEND_TEXT_TXT	varchar(255)	Legend on the sign (text only) no HTLM tag
Sign Width	SSOS_EMS / SIGN_WIDTH_NBR	decimal(8,2)	Sign Width
Sign Height Number	SSOS_EMS / SIGN_HEIGHT_NBR	decimal(8,2)	Sign Height
Square Foot	SSOS_EMS / SQ_FT_NBR	decimal(8,2))	Square Foot
Sign Cost	SSOS_EMS / SIGN_COST_NBR	decimal(8,2)	Sign Cost
Notes	SSOS_EMS / NOTES_TXT	varchar(255)	Notes on the sign
Substrate	SSOS_EMS / SUBSTRATE_TXT	varchar(6)	Substrate type of the sign
Sheet	SSOS_EMS / SHEET_TXT	varchar(15)	Sheeting type of the sign
Background Color	SSOS_EMS / BACKGROUND_COLOR_TXT	varchar(21)	Background color of the sign
Applied Color	SSOS_EMS / APPLIED_COLOR_TXT	varchar(41)	Applied color of the sign
Is Permanent	SSOS_EMS / IS_PERMANENT_IND	varchar(5)	Is Permanent indicator
Warehouse Location	SSOS_EMS / WAREHOUSE_LOCATION_TXT	varchar(31)	Warehouse location of the warehouse signs

Production Method	SSOS_EMS / PRODUCTION_METHOD_TXT	varchar(15)	Production method of the sign
Screen Location	SSOS_EMS / SCREEN_LOCATION_NBR	numeric(4)	Screen Location of the sign
Screen Condition	SSOS_EMS / SCREEN_CONDITION_TXT	varchar(15)	Screen condition of the sign
Screen Frame Size	SSOS_EMS / SCREEN_FRAME_SIZE_TXT	varchar(15)	Screen Frame Size
Sign Positive Location	SSOS_EMS / SIGN_POSITIVE_LOC_TXT	varchar(5)	Sign Positive Location
Design Date	SSOS_EMS / DESIGN_DT	datetime	Design Date
Specification	SSOS_EMS / SPECS_TXT	varchar(5)	Specification of the sign
Open Size	SSOS_EMS / OPEN_SIZE_TXT	varchar(31)	Open Size of the Sign
Sign Cost Per Square Foot	SSOS_EMS / SIGN_COST_SQFT_NBR	decimal(8,2)	Sign Cost Per Square Foot
Bypass indicator	SSOS_EMS / BY_PASS_IND	varchar(15)	Bypass indicator indicates whether this sign should bypass Central Office or not
Applied Color Number	SSOS_EMS / NUM_APPLIED_COL_NUM	varchar(5)	Applied Color Number of the sign

PACKAGES

SSOS_PACKAGES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Package Number*	SSOS_PACKAGES / PACKAGE_NBR	numeric(6)	Unique number assigned to each Package in SSOS
Package Number	SSOS_PACKAGES / PACKAGE_NUM	varchar(15)	Unique number assigned to each package in SSOS. This number consists of T + 2-digit district numbers + '-' + 4 digit sequence number per district per fiscal year + '-' + 2- digit fiscal year. E.g. T04-0012-02 is package no. 12 in year 2002 for district 4
Package Sequence Number	SSOS_PACKAGES / PACKAGE_SEQ_NBR	numeric(6)	Package sequence per district per fiscal year
Notes	SSOS_PACKAGES / NOTES_TXT	varchar(255)	Notes on the package
Created By	SSOS_PACKAGES / CREATED_BY_TXT	varchar(31)	User name of the user who created the package
District Number	SSOS_PACKAGES / DISTRICT_NBR	smallint	District number where the package was packed to
Status	SSOS_PACKAGES / STATUS_STTS	varchar(31)	Status of the package
Packed Date	SSOS_PACKAGES / PACKED_DT	datetime	Time and date on which the package was created

SSOS_PACKAGE_ITEMS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Package Item Number*	SSOS_PACKAGE_ITEMS / PACKAGE_ITEM_NBR	numeric(6)	Unique number assigned to each Package Item in SSOS
Package Number	SSOS_PACKAGE_ITEMS / PACKAGE_NBR	numeric(6)	Reference number to Package table
Order Number	SSOS_PACKAGE_ITEMS / ORDER_NBR	numeric(6)	Reference number to Order Table
Quantity Packed	SSOS_PACKAGE_ITEMS / QTY_PACKED_AMT	numeric(4)	Number of signs in the package item

PRODUCTIONS

SSOS_PRODUCTIONS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Production Number*	SSOS_PRODUCTIONS / PRODUCTION_NBR	numeric(6)	Unique number assigned to each Production in SSOS
Planning Date	SSOS_PRODUCTIONS / PLANNING_DT	datetime	Time and date on which the production was created
Notes	SSOS_PRODUCTIONS / NOTES_TXT	varchar(255)	Note on the production
Created By	SSOS_PRODUCTIONS / CREATED_BY_TXT	varchar(31)	User name of the user who created the production
Status	SSOS_PRODUCTIONS / STATUS_STTS	varchar(15)	Status of the production
Priority	SSOS_PRODUCTIONS / PRIORITY_STTS	varchar(31)	Priority of the production
Type	SSOS_PRODUCTIONS / TYPE_STTS	varchar(31)	Type of the production e.g. Silk Screen, Copy By Hand, etc…
Contact Person	SSOS_PRODUCTIONS / CONTACT_PERSON_TXT	varchar(63)	The name of the person who is in charge for this production
Finishing Date	SSOS_PRODUCTIONS / FINISHING_DT	datetime	Time and date when the production status changed to finished

SSOS_PRODUCTION_ITEMS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Production Item Number*	SSOS_PRODUCTION_ITEMS / PRODUCTION_ITEM_NBR	numeric(6)	Unique number assigned to each Production Item in SSOS
Production Number	SSOS_PRODUCTION_ITEMS / PRODUCTION_NBR	numeric(6)	Reference to production table
Order Number	SSOS_PRODUCTION_ITEMS / ORDER_NBR	numeric(6)	Reference to order table
Production Quantity	SSOS_PRODUCTION_ITEMS / PRODCUTION_QTY_AMT	numeric(4)	Number of signs in the production item

ATTRIBUTES VALUES

SSOS_ORDER_TYPES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Order Type Number*	SSOS_ORDER_TYPES / ORDER_TYPE_NBR	numeric(2)	Unique number assigned to each Order Type in SSOS
Order Type Text	SSOS_ORDER_TYPES / ORDER_TYPE_TXT	varchar(63)	Order Type
Abbreviation	SSOS_ORDER_TYPES / ABBREVIATION_TXT	varchar(15)	Order Type Abbreviation
Default Indicator	SSOS_ORDER_TYPES / DEFAULT_IND	varchar(15)	Indicates whether this is a default record
Sequence Number	SSOS_ORDER_TYPES / SEQ_NBR	numeric(2)	Sequence number used to order the records in the table

SSOS_MATERIAL_TYPES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Material Type Number*	SSOS_MATERIAL_TYPES / MATERIAL_TYPE_NBR	numeric(2)	Unique number assigned to each Material Type in SSOS
Material Type Text	SSOS_MATERIAL_TYPES / MATERIAL_TYPE_TXT	varchar(63)	Material Type
Abbreviation	SSOS_MATERIAL_TYPES / ABBREVIATION_TXT	varchar(15)	Material Type Abbreviation
Default Indicator	SSOS_MATERIAL_TYPES / DEFAULT_IND	varchar(15)	Indicates whether this is a default record
Sequence Number	SSOS_MATERIAL_TYPES / SEQ_NBR	numeric(2)	Sequence number used to order the records in the table

SSOS_SHEETING_TYPES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Sheeting Type Number*	SSOS_SHEETING_TYPES / SHEETING_TYPE_NBR	numeric(2)	Unique number assigned to each Sheeting Type in SSOS
Sheeting Type Text	SSOS_SHEETING_TYPES / SHEETING_TYPE_TXT	varchar(63)	Sheeting Type
			Sheeting Type Abbreviation

Abbreviation	SSOS_SHEETING_TYPES / ABBREVIATION_TXT	varchar(15)	
Default Indicator	SSOS_SHEETING_TYPES / DEFAULT_IND	varchar(15)	Indicates whether this is a default record
Sequence Number	SSOS_SHEETING_TYPES / SEQ_NBR	numeric(2)	Sequence number used to order the records in the table

SSOS_SUBSTRATE_TYPES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Substrate Type Number*	SSOS_SUBSTRATE_TYPES / SUBSTRATE_TYPE_NBR	numeric(2)	Unique number assigned to each Substrate Type in SSOS
Substrate Type Text	SSOS_SUBSTRATE_TYPES / SUBSTRATE_TYPE_TXT	varchar(63)	Substrate Type
Abbreviation	SSOS_SUBSTRATE_TYPES / ABBREVIATION_TXT	varchar(15)	Substrate Type Abbreviation
Default Indicator	SSOS_SUBSTRATE_TYPES / DEFAULT_IND	varchar(15)	Indicates whether this is a default record
Sequence Number	SSOS_SUBSTRATE_TYPES / SEQ_NBR	numeric(2)	Sequence number used to order the records in the table

SSOS_AGENCIES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Agency Number*	SSOS_AGENCIES / AGENCY_NBR	numeric(3)	Unique number assigned to each Agency in SSOS
Agency Text	SSOS_AGENCIES / AGENCY_TXT	varchar(63)	Agency Full Name
Abbreviation	SSOS_AGENCIES / ABBREVIATION_TXT	varchar(15)	Agency Name Abbreviation
Default Indicator	SSOS_AGENCIES / DEFAULT_IND	varchar(15)	Indicates whether this is a default record
Sequence Number	SSOS_AGENCIES / SEQ_NBR	numeric(3)	Sequence number used to order the records in the table

QUERIES

SSOS_QUERY_TABLES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Query Number*	SSOS_QUERY_TABLES / QUERY_NBR	numeric(4)	Unique number assigned to each Query in SSOS
Query Name	SSOS_QUERY_TABLES / QUERY_NAME_TXT	varchar(31)	Query name
District Number	SSOS_QUERY_TABLES / DISTRICT_NBR	smallint	District Number who created the query
Created By	SSOS_QUERY_TABLES / CREATED_BY_TXT	varchar(31)	User Name of the user who created the query
SQL Statement	SSOS_QUERY_TABLES / SQL_STATEMENT_TXT	varchar(511)	SQL statement that define the query
Creation Date	SSOS_QUERY_TABLES / CREATION_DT	datetime	Time and date when the query was created

SSOS_QUERY_FIELDS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Field Name*	SSOS_QUERY_FIELDS / FIELD_NAME_TXT	varchar(31)	Unique number assigned to each Field Name in SSOS
Column Name	SSOS_QUERY_FIELDS / COLUMN_NAME_TXT	varchar(31)	Name of the column that has the look up values
Data Type	SSOS_QUERY_FIELDS / DATA_TYPE_TXT	varchar(31)	Data type of the field
Table Name	SSOS_QUERY_FIELDS / TABLE_NAME_TXT	varchar(31)	The table that holds the look up values
Table Abbreviation	SSOS_QUERY_FIELDS / TABLE_ABBREVIATION_TXT	varchar(5)	Abbreviation of the table name

REFERENCE

DISTRICT_COUNTIES

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
District Number*	DISTRICT_COUNTIES / DISTRICT_NBR	smallint	District Number
County Abbreviation Code	DISTRICT_COUNTIES / COUNTY_ABREV3_CD	char(3)	3-characters abbreviation associated with the district number

EMAIL

SSOS_EMAILS

"PRETTY" NAME	TABLE NAME / FIELD NAME	DATA TYPE	DESCRIPTION
Ssos Group*	SSOS_EMAILS / DISTRICT_NBR	Varchar(31)	User Group
District Number	SSOS_EMAILS / DISTRICT_NBR	Small	District Number
Email	SSOS_EMAILS / EMAIL_TXT	Varchar(255)	Email addresses where the notification will be sent to
Deleted	SSOS_EMAILS / DELETED_IND	Varchar(15)	Email will be sent when order status become Deleted
Pending	SSOS_EMAILS / PENDING_IND	Varchar(15)	Email will be sent when order status become Pending
Approved	SSOS_EMAILS / APPROVED_IND	Varchar(15)	Email will be sent when order status become Approved
Archived	SSOS_EMAILS / ARCHIVED_IND	Varchar(15)	Email will be sent when order status become Archived
Received	SSOS_EMAILS / RECEIVED_IND	Varchar(15)	Email will be sent when order status become Received
Completed	SSOS_EMAILS / COMPLETED_IND	Varchar(15)	Email will be sent when order status become Completed
Delivered	SSOS_EMAILS / DELIVERED_IND	Varchar(15)	Email will be sent when order status become Delivered
Submitted	SSOS_EMAILS / SUBMITTED_IND	Varchar(15)	Email will be sent when order status become Submitted
In Production	SSOS_EMAILS / IN_PRODUCTION_IND	Varchar(15)	Email will be sent when order status become In Production

Table 4.1 Database Design Tables

4.2 Business Logic Implementation

After the database schema and UML design are finalized, the business logics are implemented according to the design. SSOS is implemented in Java as object-oriented classes. Those classes will be instantiated as needed and the instantiated objects or array of objects will be used in each transaction as parameters passed. Follow the J2EE architecture, each database table will have a corresponding bean class. A bean class has aligned data attributes and setter and getter methods for storing and retrieving the data attributes. These methods manage the data transfer between the database and the program. The descriptions of the bean classes in SSOS can be found in the SSOS CD under the ssos_javadoc directory (starting with index.html).

4.3 User Interface Implementation

User interfaces of SSOS are implemented in JSP with Java Script support for dynamic user interaction. As introduced in Chapter I, JSP is a server side program that will be compiled into a Java class when called from a client. Different from CGI and ASP, JSP will be compiled once at the first access and will be loaded into the Java virtual machine for future referencing. This feature largely reduces the turnaround time when multiple accesses happen. JSP files in SSOS can be found in the SSOS CD under the ssos directory. The purpose of each is described in the following tables.

4.3.1 Root JSP Files

Item	File	Category	Summary
1	"authenticate.jsp"	root	Authentication based on database stored users (for testing purpose only)
2	"dologin.jsp"	root	Authentication based on OneLogin security system (for final deployment)
3	"login.jsp"	root	Welcome page for entering "username" and "password"
4	"logout.jsp"	root	Logout the user and close the session

4.3.2 District – District Module Files

Item	File	Category	Summary
1	"adddesign.jsp"	District	Add new legend design with HTML tags and pictures
2	"addhistory.jsp"	District	Add new customized record in the order history table
3	"addhtml.jsp"	District	Add HTML design to the overall legend design
4	"addtext.jsp"	District	Add text design to the overall legend design
5	"changeall.jsp"	District	Change the status of all checked orders
6	"changestatus.jsp"	District	Change the status of a single order
7	"createquery.jsp"	District	Create customized query and save it in the database
8	"deletequery.jsp"	District	Delete saved query from the database
9	"design.jsp"	District	List sign attributes ready to make order
10	"design1.jsp"	District	List sign attributes ready to make order with price management
11	"design2.jsp"	District	List sing attributes ready to make order without price management
12	"doaction.jsp"	District	Actions (add line, add image, change alignment) for customized HTML legend
13	"ems.jsp"	District	Search for signs by typing few digits of the EMS number
14	"legend.jsp"	District	Search for signs in the database by providing few letters of the legend

15	"legenddesign.jsp"	District	View the HTML legend design
16	"makeorder.jsp"	District	Make an order and add it to the database
17	"moveline.jsp"	District	Change the alignment of a line
18	"nestedorder.jsp"	District	Order orders in the vieworder.jsp page based on multi columns
19	"productionmethods.jsp"	District	Edit/View production methods in pricing table
20	"queryordernumber.jsp"	District	Look up an order by giving its order number
21	"savequery.jsp"	District	Save a query in the database
22	"sheeting.jsp"	District	Search for signs by typing few digits of the EMS number
23	"signcode.jsp"	District	Search for signs by typing few digits of the Sign Code
24	"signproductionreportbean.jsp"	District	Generate a report for a sign production
25	"signproductionreportbean2.jsp"	District	Generate a report for a sign production
26	"substrate.jsp"	District	Pricing table entry for substrate type
27	"updateorder.jsp"	District	Apply order modifications in the database
28	"updatepm.jsp"	District	Change price table entry for Production Method
29	"updatesheeting.jsp"	District	Change price table entry for sheeting
30	"updatest.jsp"	District	Change price table entry for substrate
31	"uploadcad.jsp"	District	Upload a CAD file to the server
32	"uploadimage.jsp"	District	Upload a sign image to the server
33	"viewcad.jsp"	District	View an attached CAD file
34	"viewhistory.jsp"	District	View the history of an order
35	"viewimage.jsp"	District	View an attached sign image
36	"viewmodify.jsp"	District	View order details in the modify mode
37	"vieworders.jsp"	District	View summery of a list of orders
38	"viewqueries.jsp"	District	View a list of saved queries
39	"viewreport.jsp"	District	View saved reports
40	"viewreport2.jsp"	District	View saved reports

4.3.3 Central Module JSP Files

Item	File	Category	Summary
1	"adddesign.jsp"	Central	Add new legend designs with HTML tags and pictures
2	"addhistory.jsp"	Central	Add new customized records in the order history table
3	"addhtml.jsp"	Central	Add HTML designs to a legend design
4	"addtext.jsp"	Central	Add text designs to a legend design
5	"bypass.jsp"	Central	List all bypass conditions
6	"changeall.jsp"	Central	Change the status of a group of orders
7	"changestatus.jsp"	Central	Change the status of a single order
8	"createbypass.jsp"	Central	Add new bypass conditions
9	"createquery.jsp"	Central	Add new query definitions to the database
10	"deletebypass.jsp"	Central	Delete bypass conditions
11	"deletequery.jsp"	Central	Delete a saved query from the database
12	"doaction.jsp"	Central	Actions (add line, add image, change alignment) for a customized HTML legend
13	"executebypass.jsp"	Central	Run bypass conditions
14	"legenddesign.jsp"	Central	View HTML legend designs
15	"moveline.jsp"	Central	Change the alignment of a line
16	"nestedorder.jsp"	Central	Order the orders in the vieworder.jsp page based on multi columns
17	"queryordernumber.jsp"	Central	Lookup an order based on its order number
18	"savebypass.jsp"	Central	Save bypass conditions in the database
19	"savequery.jsp"	Central	Save query definitions in the database
20	"updateorder.jsp"	Central	Apply order modifications in the database
21	"uploadcad.jsp"	Central	Upload CAD files to the server
22	"uploadimage.jsp"	Central	Upload a sign image to the server
23	"viewcad.jsp"	Central	View an attached CAD file
24	"viewhistory.jsp"	Central	View the history of an order
25	"viewimage.jsp"	Central	View an attached sign image
26	"viewmodify.jsp"	Central	View order details in the modify mode
27	"vieworders.jsp"	Central	View summery of a list of orders
28	"viewqueries.jsp"	Central	View a list of saved queries

4.3.4 Planning Module JSP Files

Item	File	Category	Summary
1	"adddesign.jsp"	planning	Add new legend designs with HTML tags and pictures
2	"addhistory.jsp"	planning	Add new customized records in the order history table
3	"addhtml.jsp"	planning	Add HTML designs to the overall legend design
4	"addtext.jsp"	planning	Add text designs to the overall legend design
5	"changestatus.jsp"	planning	Change the status of an order
6	"closeproduction.jsp"	planning	Change the status of the production to "finished" and change the status of orders inside that production to "completed"
7	"createquery.jsp"	planning	Add new query definitions to the database
8	"deletequery.jsp"	planning	Delete saved queries from the database
9	"designscreen.jsp"	planning	View the printout screens for production planning
10	"doaction.jsp"	planning	Actions (add line, add image, change alignment) for the customized HTML legend
11	"finishingscreen.jsp"	planning	View the printout screens for finishing production
12	"legenddesign.jsp"	planning	View an HTML legend design
13	"makeproduction.jsp"	planning	Save a production in the database
14	"moveline.jsp"	planning	Change the alignment of a line
15	"mvproduction.jsp"	planning	Confirmation message before creating a production
16	"nestedorder.jsp"	planning	Order the orders in the vieworder.jsp page based on multi columns
17	"queryordernumber.jsp"	planning	Lookup an order based on its order number
18	"savequery.jsp"	planning	Save a query definition in the database
19	"silkscreen.jsp"	planning	View the printout screens for silkscreen production
20	"updateorder.jsp"	planning	Apply order modifications in the database
21	"uploadcad.jsp"	planning	Upload a CAD file to the server
22	"uploadimage.jsp"	planning	Upload a sign image to the server
23	"viewcad.jsp"	planning	View an attached CAD file
24	"viewhistory.jsp"	planning	View the history of an order
25	"viewimage.jsp"	planning	View an attached sign image
26	"viewmodify.jsp"	planning	View order details in the modify mode
27	"vieworders.jsp"	planning	View summery of a list of orders

28	"viewproductions.jsp"	planning	View a list of productions
29	"viewqueries.jsp"	planning	View a list of saved queries

4.3.5 Packing Module JSP Files

Item	File	Category	Summary
1	"addhistory.jsp"	packing	Add new legend designs with HTML tags and pictures
2	"createquery.jsp"	packing	Add new query definitions to the database
3	"deletequery.jsp"	packing	Delete saved queries from the database
4	"makepackage.jsp"	packing	Save a new package in the database
5	"mvpackage.jsp"	packing	Confirm the creation of a package before saving it in the database
6	"nestedorder.jsp"	packing	Order the orders in the vieworder.jsp page based on multi columns
7	"queryordernumber.jsp"	packing	Lookup an order based on its order number
8	"querypacking.jsp"	packing	Search for packages using certain parameters
9	"savequery.jsp"	packing	Save a query definition in the database
10	"viewhistory.jsp"	packing	View the history of an order
11	"viewmodify.jsp"	packing	View order details in the modify mode
12	"vieworders.jsp"	packing	View summery of a list of orders
13	"viewpackages.jsp"	packing	View summery of a list of packages
14	"viewqueries.jsp"	packing	View a list of saved queries

4.3.6 Transfer Module JSP Files

Item	File	Category	Summary
1	"addhistory.jsp"	transfer	Add new legend designs with HTML tags and pictures
2	"createquery.jsp"	transfer	Add new query definitions to the database
3	"deletequery.jsp"	transfer	Delete a saved query from the database
4	"nestedorder.jsp"	transfer	Order the orders in the vieworder.jsp page based on multi columns
5	"queryordernumber.jsp"	transfer	Lookup an order based on its order number
6	"querypacking.jsp"	transfer	Lookup a package based on its package number

7	"savequery.jsp"	transfer	Save query definitions in the database
8	"transferpackage.jsp"	transfer	Change the status of a package to "transferred"
9	"viewhistory.jsp"	transfer	View the history of an order
10	"viewmodify.jsp"	transfer	View order details in the modify mode
11	"vieworders.jsp"	transfer	View the summery of a list of orders
12	"viewpackages.jsp"	transfer	View the summery of a list of packages
13	"viewqueries.jsp"	transfer	View a list of saved queries

4.4 Security Implementation

Security is a very important issue in web development. Normally there are two approaches: system security and application security. At the previous attempt of SSOS prototype, system security with Tomcat was used. Using system security has a key drawback—the impairing of program mobility. The application must and will only be secure in the system where the security script was run and when the application needs to be transferred to a new system, or a new type of server, the whole security script needs to run again in that system and if that system is not compatible with the current security script, a new script is needed for system security.

Currently application security is used by SSOS based on the OneLogin security system implemented by DoIT as shown in Figure 4.2. Each SSOS user will have one or more groups assigned to him/her. The first group will be the default one and the others will be on the toolbar so he/she can switch between them. If an user failed to pass the OneLogin security system he/she will be returned to the login page.

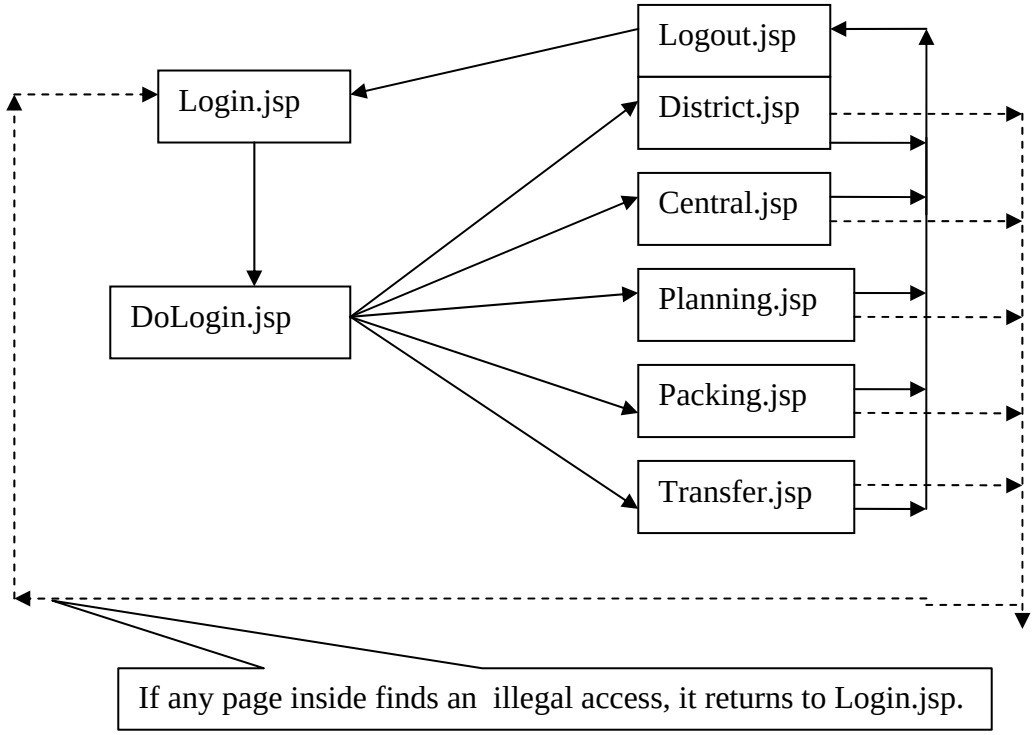

Figure 4.2 Security Design

4.5 System Deployment

System deployment is done in two phases first phase is the database deployment in which the database tables are create and template data are inserted into the tables. The second phase is deploying the jsp files and class files in the web server.

4.5.1 Database Deployment

Database deployment is done by creating the tables, run ddl/CreateTables.sql file which will create the necessary tables for SSOS. Then run ddl/looupdate.sql file, this file will insert the lookup values and other template data in the database tables. Finally run ddl/ems.sql file, this file

will insert the sign information in the database this will take about 10 minutes since it will insert about 5,000 records in the database.

4.5.2 JSP and Class Files Deployment

Before copying the jsp files and the class files to the tomcat directory few changes maybe necessary:

- In SSOS CD under ssos_src\ssos\database\config\ directory ConfigManager.java file contain references to
    1. webHome
    2. DB driver
    3. DB Url
    4. dbUserName/dbPassword
    5. SMTP server
    6. System administrator email address

    These variables should be changed to the corresponding values in the deployment environment.
- Any reference for district_counties should be replace with odotref.dbo.district_county these references may be found in
    1. In SSOS CD ssos_src\ssos\database\manager\DBObjectManager.java
    2. In SSOS CD ssos\packing\querypacking.jsp
    3. In SSOS CD ssos\transfer\querypacking.jsp

- In SSOS CD all directories under ssos\districtcad and ssos\districtimage need to be deleted. Note do not delete the directory itself.
- The form target in SSOS CD ssos\login.jsp should target dologin.jsp
- All references in SSOS CD ssos_src\ssos\database\config\ConfigManager.java to HTML_LEGEND_TEXT should be changed to HTML_LEGEND_TXT

After making the above changes. Recompile the java code and copy it along with the jsp file to the web server. The web server needs to be restarted after every new deployment.

CONCLUDING REMARKS

The project developed a smart sign ordering system (SSOS) to assist in the sign ordering process. Specifically, the SSOS program has built into it automated functions for data entry during preparation of sign orders, and provides on-line data review and modification capabilities in the sign ordering and validation process. In addition, it enables querying and sorting, and helps tracking the orders in the production and delivery phases. The system improves the work efficiency of the sign ordering process, reduces human errors in order handling, speeds up review time, and aligns order submissions better with the current sign production method.

As the name suggests, SSOS is designed primarily for sign ordering management. To make the system more useful, such as facilitating sign production management in the Sign Shop, further development is needed to expand its functional features. The project team is currently discussing with the Sign Shop about such needs.

BIBLIOGRAPHY

1. Sun java tutorial (java.sun.com/docs/books/tutorial/).
2. Sun Java Servlet Introduction page (http://java.sun.com/products/servlet/).
3. Sun J2EE tutorial (java.sun.com/j2ee/tutorial/).
4. UML Distilled, A Brief Guide to the Standard Object Modeling Language (2nd Edition), Martin Fowler, Kendall Scott, 2000.
5. Smart Sign Ordering System, Part III Sign Shop. Yongbin Ma, Masters Project Report, The University of Akron, 2001.
6. Smart Sign Ordering System, Part II Central Office. Bin Yang, Masters Project Report, The University of Akron, 2001.