

# Online Shopping System based on WAP

Tao Zhou

Kgs. Lyngby 2004  
IMM-THESIS-2004-90

Technical University of Denmark  
Informatics and Mathematical Modelling  
Building 321, DK-2800 Lyngby, Denmark  
Phone +45 45253351, Fax +45 45882673  
[reception@imm.dtu.dk](mailto:reception@imm.dtu.dk)  
[www.imm.dtu.dk](http://www.imm.dtu.dk)

IMM-THESIS: ISSN 1601-233X

# Table of Content

1	Introduction.....	- 4 -
1.1	Definitions and key words .....	- 4 -
1.2	Motivation and Goals.....	- 6 -
1.2.1	Motivation, WAP overview .....	- 6 -
1.2.2	Goals .....	- 8 -
1.3	Correlated Work.....	- 8 -
1.4	About this thesis.....	- 9 -
2	Problem Domain .....	- 11 -
2.1	Problem.....	- 11 -
2.2	Users classes .....	- 12 -
2.3	Functional Requirements .....	- 13 -
2.3.1	The expectative layout of WAP shop (Visualization functionalities) .....	- 13 -
2.3.2	The expectative customer shopping process (Ordering functionalities).....	- 15 -
2.3.3	The expectative layout of Administrative Module (Visualization functionalities) .....	- 16 -
2.3.4	The expectative management process (Editing functionalities) .....	- 17 -
2.3.5	Functional requirements of Database system .....	- 18 -
2.4	Introduction of correlation between functionalities and system's files .....	- 19 -
2.5	Correlated work and conclusion .....	- 20 -
3	Development Tools.....	- 21 -
3.1	Introduction of development tools used in WOSS system .....	- 21 -
3.2	Using WML .....	- 23 -
3.3	Using MySQL.....	- 25 -
3.4	Using Orion Application server .....	- 28 -
3.4.1	JavaBeans overview.....	- 28 -
3.4.2	Orion Application server.....	- 29 -
3.5	Using JSP .....	- 32 -
3.5.1	JSP overview.....	- 32 -
3.5.2	JSP configuration.....	- 32 -
3.5.3	JSP syntax and simple examples of WOSS .....	- 33 -
3.6	Using JDBC .....	- 35 -
3.7	Using JavaScript and HTML .....	- 39 -
3.7.1	HTML DOM event .....	- 39 -
3.7.2	Create a table.....	- 40 -
3.7.3	DOM checkbox.....	- 40 -
3.7.4	HTML input form .....	- 40 -
3.7.5	Dropdown list in a form.....	- 41 -
3.8	Selecting Operating System.....	- 41 -
3.9	Conclusion .....	- 41 -
4	System Design .....	- 42 -
4.1	Overview.....	- 42 -
4.2	WOSS system structure .....	- 42 -
4.3	Design of Administrative module (Web server).....	- 45 -
4.4	Design of Customer Interface .....	- 46 -

4.5 Design of Database management system.....	- 49 -
4.6 Conclusion .....	- 53 -
5 System Implementation .....	- 54 -
5.1 Implementation of Administrative module (Web server).....	- 54 -
5.1.1 Implementation of ExtraWapImage.jsp (WAP SHOP EDIT) .....	- 55 -
5.1.2 Implementation of PageEdit.jsp (Page Editor) .....	- 58 -
5.1.3 Implementation of data query and retrieving the data to display by using JavaScript and SQL.....	- 63 -
5.1.4 Implementation of JDBC .....	- 65 -
5.2 Implementation of Customer interface .....	- 67 -
5.2.1 Implementation of wapShop.jsp and extraview.jsp .....	- 68 -
5.2.2 Implementation of Customer ordering.....	- 72 -
5.3 Implementation MySQL database .....	- 74 -
5.4 Conclusion .....	- 76 -
6 System Test.....	- 77 -
6.1 Component test .....	- 77 -
6.1.1 Component test of Customer Interface .....	- 77 -
6.1.2 Component Test of Administrative Module .....	- 79 -
6.2 Integration test .....	- 80 -
6.2.1 Integration test of editing functionalities .....	- 81 -
6.2.2 Integration test of ordering functionalities and data saved in database .....	- 84 -
6.3 Conclusion .....	- 85 -
7 Conclusion and proposal.....	- 87 -
7.1 Achievements.....	- 87 -
7.2 Future possible improvements of WOSS system.....	- 88 -
8 Bibliography and Appendixes .....	- 89 -
8.1 Bibliography .....	- 89 -
8.2 Source code.....	- 90 -
8.2.1 ExtraWapImage.jsp.....	- 90 -
8.2.2 ewiupdate.jsp .....	- 93 -
8.2.3 PageEdit.jsp .....	- 95 -
8.2.4 ewidel.jsp .....	- 100 -
8.2.5 ewipageupdate.jsp.....	- 101 -
8.2.6 picupload.jsp .....	- 102 -
8.2.7 wapShop.jsp.....	- 107 -
8.2.8 extraview.jsp .....	- 107 -
8.2.9 order.jsp .....	- 109 -
8.2.10 saveOrder.jsp .....	- 110 -
8.2.11 orderDel.jsp.....	- 111 -
8.2.12 Connloader.java .....	- 112 -

## List of Figures

Figure 1.1 UML model of WAP e-commerce .....	- 8 -
Figure 2.1 Prototype of Online Shopping System based on WAP .....	- 12 -
Figure 2.2 UML diagram of WAP shop layout .....	- 14 -
Figure 2.3 Prototype of Administrative module .....	- 17 -
Figure 3.1 WOSS system structure .....	- 22 -
Figure 3.2 Sketch of the Orion application server .....	- 29 -
Figure 3.3 JSP Architecture .....	- 33 -
Figure 3.4 Java.sql tree .....	- 36 -
Figure 4.1 The arrangement of JSP files in Customer Interface .....	- 43 -
Figure 4.2 The arrangement of JSP files in Administrative Module .....	- 43 -
Figure 4.3 WOSS system structure .....	- 44 -
Figure 4.4 Correlations of different JSP files .....	- 48 -
Figure 4.5 E/R graphical analysis of data structures in WOSS system .....	- 52 -
Figure 5.1 Screenshot of WAP SHOP EDIT homepage .....	- 55 -
Figure 5.2 Screenshot of Page Editor for product catalogs .....	- 58 -
Figure 5.3 Screenshot of Page Editor for products .....	- 60 -
Figure 5.4 Screenshot of Picture Uploading .....	- 63 -
Figure 5.5 Screenshot of Customer shopping page .....	- 68 -
Figure 5.6 Screenshot of the product catalog 'Sports' .....	- 69 -
Figure 5.7 Screenshot of the product 'Tent' .....	- 69 -
Figure 5.8 Screenshot of Customer Ordering .....	- 73 -
Figure 6.1 Component Test of wapShop homepage by using OpenWave .....	- 78 -
Figure 6.2 Component Test of product page by using OpenWave .....	- 78 -
Figure 6.3 Component Test of Customer Ordering by using OpenWave .....	- 79 -
Figure 6.4 Test of uploading the picture (1) .....	- 80 -
Figure 6.5 Test of uploading pictures (2) .....	- 80 -
Figure 6.6 Page Editor after adding 'TEST' .....	- 81 -
Figure 6.7 Shopping page after adding 'TEST' .....	- 82 -
Figure 6.8 Page Editor after adding 'Girls Picture' .....	- 82 -
Figure 6.9 Page Editor after uploading the picture .....	- 82 -
Figure 6.10 Shopping page after uploading the picture .....	- 83 -
Figure 6.11 Page Editor after adding the price .....	- 83 -
Figure 6.12 Shopping page after adding the price .....	- 84 -
Figure 6.13 Integration test of saving customer's order .....	- 85 -
Figure 6.14 Integration test of saving customer's order in database .....	- 85 -

# 1 Introduction

In my thesis, I propose an Online Shopping System which is based on the 2.5G wireless communication technology, WAP (Wireless Application Protocol). Not only can it be implemented in office or home PCs, but also can it be compatible with cell phones which support WAP (2.5G application).

A general approach of the prototype establishment, design and implementation, etc of this Online Shopping System based on WAP is presented, by using software engineering theories and telecommunication theories.

This chapter describes the key words, motivation, goals, correlated work and the reading guide for readers. For the reason of essentiality and complexity of the problem prototype of an Online Shopping System based on WAP, I will discuss the problem domain in Chapter 2.

## 1.1 Definitions and key words

In the following chapters, to avoid the redundant repetition of the long terms, I use the key words, for example I use WOSS, instead of giving the full name in all the following chapters. Some key words are acronyms of names of system's applications, and some are definitions or explanations of technologies. If readers face a strange acronym or development technology, please first refer to this section.

**WOSS:** Acronym of WAP based Online Shopping System. I use this acronym in the context.

**Customer Interface:** the web pages for WAP customers to browse and buy products, in other words, it is a virtual online WAP shop.

**Administrative Module / Administrative Interface:** the Web server of WOSS system used by administrators to modify the information of WAP shop.

**Item:** the aggregation of all sorts of products sold in WAP shop. Furthermore, the separate shopping items which may include several different product catalogs.

**Product Catalog:** the sort of different products, for instance, all the sport products are classified as 'sports' that is named as a product catalog.

**Products:** the goods traded in WAP shop

**wapShop:** a simulated customer interface (login interface) for system demo, which means WAP shopping item.

**WAP SHOP EDIT:** administrative interface, used for editing, modifying the information of shopping items.

**Page Editor:** administrative interface, used for editing, modifying the information of product catalogs and products.

**gtom\_wap:** the database name that saves all the information of WOSS system in MySQL server.

**WAP:** Acronym of the wireless application protocol.

**J2SDK:** The J2SE Software Development Kit (SDK)<sup>1</sup> supports creating J2SE applications. (By using version 1.4.2\_04)

**JSP:** Acronym of JavaServer Pages Technology.

**MySQL:** the most popular open source<sup>2</sup> database in the world.

**JavaBeans:** JavaBeans is a component architecture for defining reusable components in Java 2 Platform for Java. It is reusable software programs that a programmer can develop and assemble easily to create sophisticated applications.

**Orion application server:** A pure java full-featured application server which supports Java 2. This application server is more or less similar as an application container which makes data configuration system (Database) and programming implementation (Java 2) compatible, and it's easier for a developer to run a simple Enterprise JavaBeans.

**JDBC:** JDBC technology<sup>3</sup> is an API (included in both J2SE and J2EE releases) that provides cross-DBMS connectivity to a wide range of SQL databases and accesses to other tabular data sources, such as spreadsheets or flat files. With a JDBC technology-enabled driver, a developer can connect all corporate data even in a heterogeneous environment.

**WML:** stands for Wireless Markup Language which is mark-up language inherited from HTML, and based on XML, designed for web pages that are displayed on a WAP browser.

**ERP:** Acronym of Enterprise Resource Planning.

---

<sup>1</sup> J2SDK, <http://java.sun.com/j2se/1.4.2/>, Sun Microsystems, 1994-2004

<sup>2</sup> MySQL, [www.mysql.com/](http://www.mysql.com/), MySQL AB, 1995-2004

<sup>3</sup> JDBC, [java.sun.com](http://java.sun.com), Sun Microsystems, 2004

**CRM:** Acronym of Customer Relationship Planning.

**Opera navigator**<sup>4</sup>: one of the most popular free- to- use web browser supporting WAP up to dated.

**OpenWave:** A mobile simulator helps developers compelling applications and services based on WAP (or other 3G wireless technologies), including XHTML, and any animation of MMS (Multimedia Messaging Service).

## 1.2 Motivation and Goals

In this section I describe the development and prospect of WAP technology that motivated me to develop an Online Shopping System based on WAP. I define the model of WAP e-commerce in this section and this model is directly associated with the prototype of Online Shopping System defined in next Chapter. In addition, the goals of this thesis, in other words, the achievements, and knowledge I want to get after WOSS system development are also presented in this section.

### 1.2.1 Motivation, WAP overview

With the development of mobile phones, the services based on mobile communication are becoming more attractive and promising than traditional Web services. According to the statistics from Nokia, there are 1.5 billion people<sup>5</sup> using mobile phones in 2004, 241 million of them are 2.5G, and 48 million are 3G. Until 2007, the number of mobile phone users will hit 2 billion, and 40% of them will subscribe mobile Internet services. This number is much more than PC users who subscribe the online services.

WAP, as 2.5G mobile communication technology, stands as the interface supporting a wireless device, such as cell phone, or PDA, to view Internet page. Earlier, users could browse text only within the very simple black and white pictures. But with the development of technology of mobile communications, for example by using WAP over GPRS networks, users can enjoy more colorful contents, such as E-mail services, online business services, E-game and entertainment, and MMS (Multimedia Message Services), etc. Currently the transmission speed of WAP over GPRS hits 20kbps - 30kbps, and it towards 3G services which has been already launched in EU countries since 1999. According to OECD.org<sup>6</sup>, in 1998, there were only 18 million WAP users in the world, and in 2002, there were 12 million WAP subscribers in EU which stand only 5.5% of 220 million mobile phone subscribers, but in 2004, 65% of mobile users subscribed WAP services, which represent over 100 million users. In Denmark, Sonofon was the first mobile operator which provided WAP services by

---

<sup>4</sup> Opera Navigator, <http://portal.opera.com/>, Opera Software ASA, 1995-2004

<sup>5</sup> Nokia News, [www.nokia.com/search/index.jsp?wsid=8&qt=news](http://www.nokia.com/search/index.jsp?wsid=8&qt=news), Nokia, Finland, 2004

<sup>6</sup> OECD: Organization for Economic Co-operation and Development, [www.oecd.org/home/](http://www.oecd.org/home/)



using Nokia Artus Messaging Platform. In 2003, there were 10,000 WAP subscribers in Denmark and it was far more than the number of 3G subscribers, when 3.DK launched its services, in 2002.

A WAP user connects to Internet through what is known as the WAP gateway. Most of mobile operators of EU have their own WAP gateways. When the WAP user browses WAP web sites, this individual cell phone sends a request to the WAP gateway, and then the request will be transferred to WAP site for the webpage a user wants. If the WAP gateway has that copy of web site, it will be compressed and sent back to the cell phone.

WAP supports most of wireless communication networks, such as GSM, CDMA, TDMA, Mobiltext, and of course it is compatible with 3G networks, such as UMTS and WCDMA. WAP is supported by most of Operating systems, such as OS/9, JavaOS, EPOC, and especially Window CE which is PDA system supporting WML.

WAP is based on existing Internet standards, such as HTML, XML, IP, and HTTP 1.1. WML is specifically devised for micro-screen devices such as cell phones or PDA with or without keyboards. This optimizes these wireless handheld devices with tiny displays and low bandwidth.

WAP as the open standard of wireless Internet communication drives the convergence of Internet services and mobile phone services. As the breakthrough of mobile e-commerce, WAP has released a new mobile Internet market force. WAP applications for mobile dispatching services improves response and efficiency, and it makes e-commerce real-time and within less resources. For instance, in WAP online shopping system, with the handsets supporting 2G (GSM), 2.5G or 3G, users can order products at the place covered by operator's services and it realizes the form, real-time, everywhere, and anytime. Another outstanding example is, in EU, the mobile devices with WAP subscription become the most common client devices for B2B, B2C applications, and according to IDC's statistics<sup>7</sup>, in the couple of years, the number of subscribers of mobile e-commerce based on WAP are more than the wired PC's accessing to e-commerce.

WAP forms a new e-commerce model with the mobile access to Internet, which is the convergence of ubiquitous clients including WAP cell phones, Web browsers, WEB/WAP servers supported by XML; WML, application servers with multi-access to database system and any GUI clients. This model actually is becoming the ubiquitous model of 2.5G and 3G mobile e-commerce. The model is visualized as Figure 1.1 by using UML<sup>8</sup>. First WAP clients connect to WEB/WAP servers that provide difference WAP services, for instance, E-mail service, Online shopping, and Information search service. Clients send data requests, for example, clients send the news request, when a WEB/WAP server receives these requests, it firstly searches

---

<sup>7</sup> Mobile eBusiness Magic White Paper, magic-SW.com, Magic, 2003, USA

<sup>8</sup> Practical UML, <http://bdn.borland.com/article/0,1410,31863,00.html#use-case-diagram>

these data locally, if it has these contents, WEB server will send them to client's WAP browsers otherwise it will apply for Application server (central server of operators). All the WAP data are saved or backedup in database.

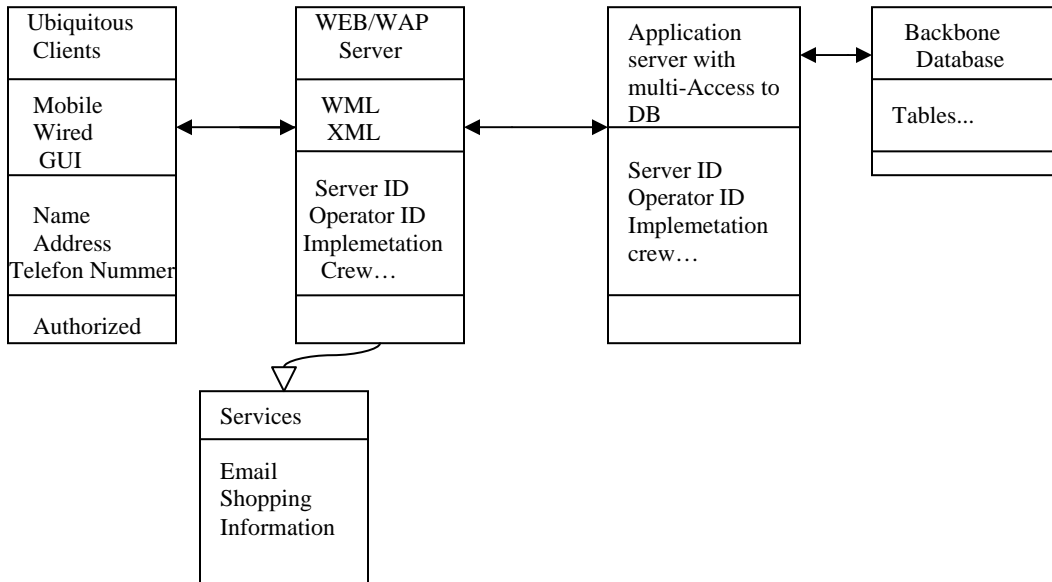


Figure 1.1 UML model of WAP e-commerce

### 1.2.2 Goals

The goals of this thesis are:

1. Study and extend the programming skills such as WML, HTML, Java, JSP and SQL syntax, etc.
2. Use Orion application server to run a typical enterprise JavaBeans and apply the theories of ERP and CRM to idealize the system.
3. Use JDBC connectivity solving communication between the control layer and the data layer.
4. Employ proper methods to implement Online shopping system, test and validate the realized system after implementation.

### 1.3 Correlated Work

Authorization system of an online system is one of important features. Without an effective authorization mechanism, the online shopping system will be vulnerable and less secure.

Basic authorization infrastructure can be implemented by database system. For example, to create a user table in database and authorize the users accessing to shopping interfaces, meanwhile, an administrator has the power to edit customer information and edit administrative interfaces. However, anyone who has the right to access into database can gather and modify these sorts of information. It will become a break-in point of the whole system.

As a result, the user interface encrypt implementation is necessary, especially WAP interface encrypt, such as the encrypt implementation of WAP shopping interface and the encrypt implementation of administrative interface. There are lots of encrypt algorithms, such as Microsoft LSA/Shell, but unfortunately it doesn't support WML syntax. As a single developer responsible for developing the complete online shopping system, I will only recommend these practical applications in this thesis, not in the workload of implementations.

#### **1.4 About this thesis**

I organize this thesis by the following chapters: Problem domain; Development tools; System design; System implementation; System test and Conclusion.

In Chapter 1, Section 1.1, I gave the most important key words of the whole thesis. These key words stand for acronyms of specific phrases of WOSS system and technologies; definitions used in system development. They will be repeatedly used in the whole thesis and it's very important for readers to read the context. Furthermore, in order to make the readers have a better understanding of this thesis, I also describe the theories and methods concerning Wireless communication technology, especially in WAP, Wireless e-commerce, especially in 2.5G and 3G mobile communications.

In Chapter 2 Problem Domain, I give the introductions of the defined prototype of Online Shopping System based on WAP, which are both in entity layouts and in conducting processes. This prototype is the guideline of the system design and the system implementation. Furthermore, to make readers easily understand the context, I give the description of correlations between functionalities and system application files, and corresponding sections in context as well, which might be, I hope, a helpful guidance for readers to go through this thesis.

In Chapter 3 Development Tools, I describe the utility of WAP, Orion Application Server, MySQL, etc in WOSS system. It includes the reason why I used these software, comparison with other software and typical syntax introductions which are only based on WOSS system development. These introductions are very important for readers to understand how I employed these tools to realize the prototype defined in Chapter 2, and to approach my goals. It is also necessary to the readers who haven't experiences with these development tools.

In Chapter 4 System design, I present the designing details of Customer Interface, WAP SHOP EDIT administrative module, and database design, such as alternatives of data types, E/R analysis, and table structure.

In Chapter 5 System implementation, I describe the implementation of WOSS system. The kernel implementations of Customer Interface, Administrative Module and MySQL database management system are presented in this Chapter.

In Chapter 6 System test, I test WOSS system by using Component test and Integration test, and a navigator of simulated WAP mobile phone, OpenWave is used.

In Chapter 7 Conclusion and proposal, I conclude on the achievements of developing WOSS system and research on this thesis.

In Chapter 8 Appendix, I list all the bibliography of this thesis and source code of WOSS system files.

## 2 Problem Domain

Online shopping system based on WAP, as its name indicates, is to provide a virtual Internet shop website to a customer who has WAP connection, mainly a WAP mobile phone to buy products, and also to provide the shop's administrator an interface to control and manipulate this website. It should be fundamentally consisted by shopping website, administrative interface and database management system. However, according to ERP theory, a complete information system should also include its CRM<sup>9</sup> system, Logistic system, Financial audit center, etc. Since in the development of WOSS system, I only focus on customer's shopping functionalities, Web server functionalities and database management functionalities, those other components of ERP theory are not covered in this thesis.

In this chapter, I define the prototype of Online Shopping System based on WAP, both in the system's structure and in the functionalities that my system should provide as well. First I give the description of the problem arising, followed by an introduction of system's user classes. Second I describe the functional requirements of Online Shopping System based on WAP, and these functionalities defined will coach the selection of development tools, system design and system implementation through. I describe the layout of WAP shop, not the technical details of design or implementation of WAP shop, but the physical layout, in other words, the commercial structure of WAP shop. I also describe a complete customer's shopping process, and a complete management process of WAP shop administrator. In addition a brief introduction of functional requirements of database system is presented. In the end, an introduction of correlations between functionalities and system files, correlated work and conclusion are presented.

The selection of development tools, system design, and system implementation coming up in the following chapters are all based on the prototype defined in this chapter.

### 2.1 Problem

Traditionally, customers are used to buy the products at the real, in other words, factual shops or supermarkets. It needs the customers to show up in the shops in person, and walk around different shopping shelves, and it also needs the owners of shops to stock, exhibit, and transfer the products required by customers. It takes labor, time and space to proceed these operations. Online shopping system based on WAP provides a solution to reduce and optimize these expenses. Authorized Customers do not need to go to the factual shops to choose, and bring the products they need by hands. They simply browse their cell phone's navigators that access to WAP shops, and evaluate the products description, pictures on the screen to choose products. In addition, the owner of WAP shops do not need to arrange, exhibit their stocked

---

<sup>9</sup> CRM: Customer Relationship Management, [www.crm2day.com](http://www.crm2day.com), CRM Today, 2001-2004

products. They just input the description, prices of products, and upload their pictures. Simply, both customers and shop owners do not need to touch the real products in the whole process of shopping, and management. In the end the logistic center will distribute the products required by customers, or products ordered by shop's owners to their locations. The payment and products' quantity will be saved in database through the data flow. These shopping, management and distribution processes greatly simplify and optimize the retail business.

According to WAP shopping process and WAP shop management process indicated above, I developed customer's shopping website (**customer interface**) corresponding to shopping process and Web server (**administrative module**) corresponding to management process. Undoubtedly, **database** management system is inevitable part of the whole shopping system. This prototype of Online Shopping System based on WAP can be visualized as the following figure. Since the distribution process/logistic system has more complicated development methods, it is not covered in this thesis.

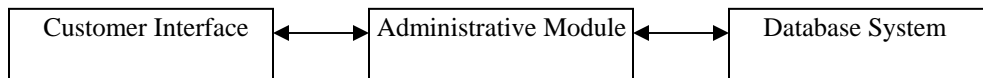


Figure 2.1 Prototype of Online Shopping System based on WAP

Customer Interface provides a virtual online shopping website for customers to buy products. Administrative module provides an interface for administrators to manipulate the customer interface, for example to modify the layout shopping website, and to edit product information. Meanwhile, the customer's data, for example, customer identifications and shopping orders are saved in database system through the connector provided by Administrative module. And edited product information is saved in database as well.

In the following section, I present the definitions of system users, for example, customers and administrators mentioned above.

## 2.2 Users classes

WOSS system is used by three kinds of users. They are users of WAP mobile phone or PC users who have the navigators supporting WAP (named as Customers), administrators of WOSS system (named as Administrators), and system maintenance crew (named as Developer, Tao Zhou). Each user has his or her attributes and privileges.

**Customers'** attributes: ID, Name, Gender, Mobile phone number, Reserve information, and Details.

Customers' privileges: Access to shopping website, and buy products.

**Administrators'** attributes: ID, Name, Gender, Mobile phone number, Reserve information, and Details.

Administrators' privileges: Access to the shopping website to check products' information, access to administrative interface (WAP SHOP EDIT) to edit products' information and customer's data.

**System maintenance crew:** ID, Name, Gender, Mobile phone number, Reserve information, and Details.

System maintenance crew's privileges: Access to shopping website to check products' information, and modify the functionality of WOSS system required by shop's owners.

All these attributes are saved in database management system as the table's fields, and I will give the explanation of them in Chapter 4 System Design.

I create a user named as '**Mike**' who has the customer's privileges, and all administrators' privileges. This designated user is only for system trial and simplifying the operation steps and 'Mike' is used not only in this chapter, but in the following chapters.

## **2.3 Functional Requirements**

In this section, according to the prototype defined above, I present the functionalities and services my system should provide to customers and administrators. First, I describe the functional requirements of Customer Interface, including visual layout of WAP shop and expectative customer shopping process. Second, I describe functional requirements of Administrative Module, including visual layout of Administrative module and expectative management process. In the end I give a brief introduction of functional requirements of database system.

### **2.3.1 The expectative layout of WAP shop (Visualization functionalities)**

Although WAP shop is a virtual Internet shopping location, it does not exist physically, customers are still used to go shopping following the traditional methods. For instance, a housewife goes into a supermarket. First the product she will buy might be beef, so she goes to butcher section, and select fresh beef. Second, she will buy some bread for breakfast, so she goes to baker section, and bring a bag of rye bread. Or for another example, Lyngby Storcenter is divided into several different retail stores, such as sport store, clothing store, and video store, etc. A well-run WAP shop should be arranged following these traditional methods, so customers can finish their shopping operation conveniently, and quickly.

As the expectative functionalities of WAP shop indicated above, first Customer Interface should provide a homepage of shopping place, for instance, a shopping center like Lyngby Storcenter. I named this place as (shopping) 'Item'. Second, single item webpage should consist of different shopping catalogs, for example a catalog including all sport products. These products catalogs are similar to different stores in Lyngby Storcenter. Third, a single product catalog should have its own

webpage to display the list of products. This is similar to the different sport products sold in the sports store. Fourth, every product should have its own webpage to display single product information including text description, pictures and price. Furthermore, the customer check-out webpage that displays the check-out choices of shopping orders should be displayed not only after product webpage but also in the item homepage after a customer finishing products ordering.

The visualization functionalities of displaying the shopping item can be deduced as the following process:

Shopping items (Homepage) --- products' catalogs (such as sports, etc.) --- Products of sports catalog (such as a water bottle) --- Details (pictures, or prices) --- Orders (buy or not, Confirmation of Order).

This layout can be visualized as the following UML diagram:

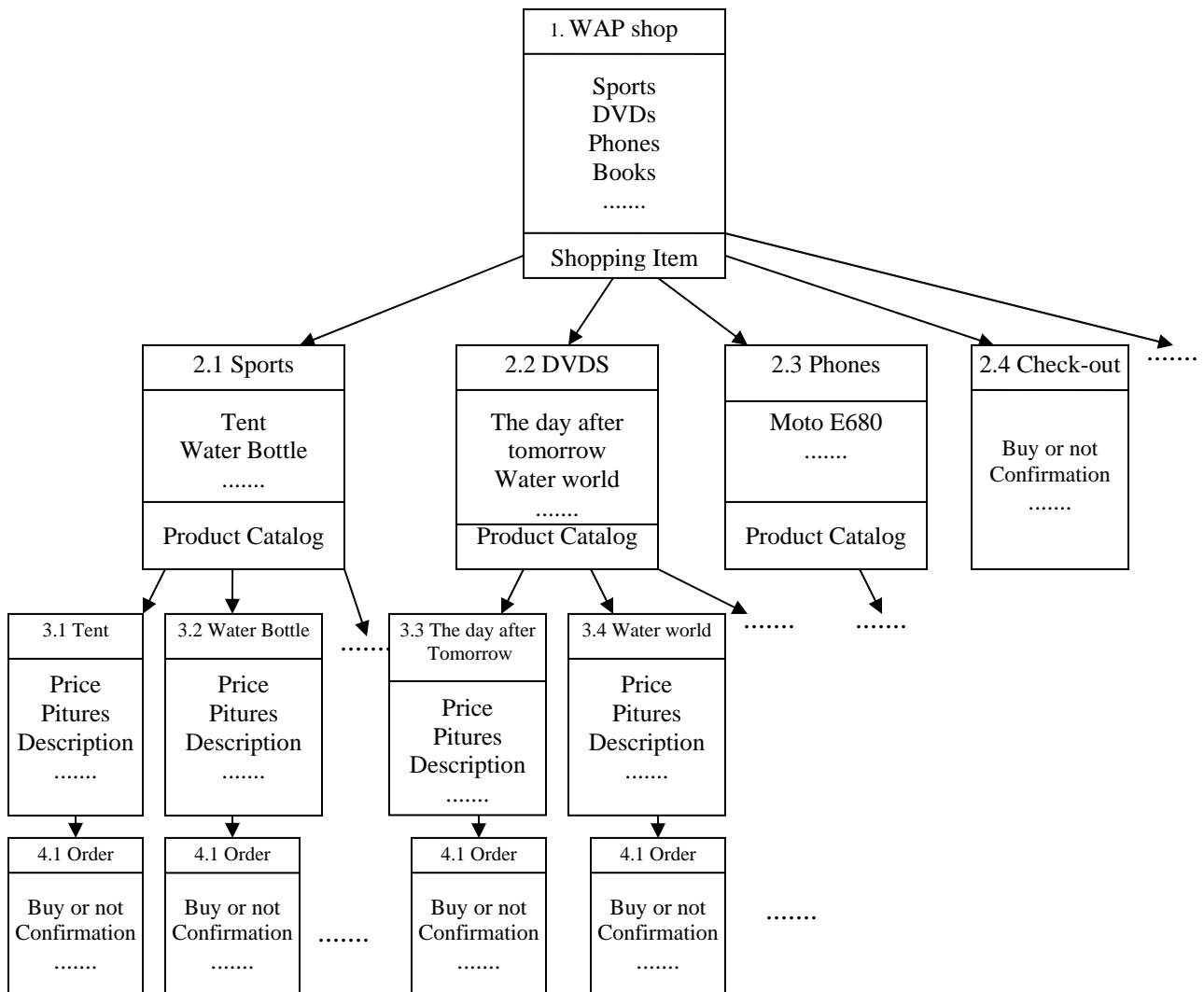


Figure 2.2 UML diagram of WAP shop layout



Undoubtedly, an item of WAP shop consists of several different shopping catalogs not only Sports, DVDs, Phones shown in figure above, Figure 2.5 is only the prototype of expectative layout of WAP shop.

### **2.3.2 The expectative customer shopping process (Ordering functionalities)**

The complete customer shopping process should begin from the customer login. It needs Customer Interface provides the functionalities of customer authorization. However, as Section 1.3 Correlated work indicated, the authorization function is not realized in WOSS system, so this functional requirement is omitted in this thesis.

Now I use the simulated customer 'Mike' to introduce the functionalities Customer interface should provide in customer shopping process. I assume Mike has been validated in WAP shop. After he goes through the shopping information, following the layout defined in previous section, Mike needs to order something for his son as gifts in WAP shop. As a result, the customer interface should not only provide the product's information to customers, but also the price. Mike can compare the different prices and decide to buy which products. Furthermore, Mike can buy more than one product in different product catalogs. It needs the customer interface to provide the functionalities of the total price calculation. And if Mike changes his mind, for example originally, he decides to buy a water bottle and a tent to his son, but finally, in the check-out webpage, he finds the total price is more than his budget. He should delete something. It needs the customer interface to provide the functionalities of orders deleting. At the end, after Mike submits his bills, it needs the customer interface to provide the functionalities of ordering confirmation to remind the customer that his bills have been recorded. All of these functionalities required above can be concluded as 'Ordering functionalities' in WOSS system.

The shopping processes of Mike can be defined as a multi-process shopping, in other words, Mike bought not only a single product. It is also possible for a customer to buy only one product and check out in that product's page

All in all, the complete customer's shopping process should be:

1. Customer login
2. Choosing the products' catalogs to enter
3. After entering a product catalog, the customer goes through the products' list
4. Enter the product's page he prefers
5. Click on 'buy now' to buy this product
6. After order saved, he can go back to homepage to reselect products
7. Repeat steps 2-6
8. Click on 'take the bill' to check out his orders, and decide if he take them all, or delete something
9. Back to homepage, logout

### **2.3.3 The expectative layout of Administrative Module (Visualization functionalities)**

Administrative module should provide a place for an administrator to modify the information of WAP shop including text descriptions, uploading pictures and price information, etc.

First, administrative module should display a homepage for an administrator to create a shopping item and this administrative homepage should display a distinguished name, for example I defined it as 'WAP SHOP EDIT' in this module. In this homepage, it should display the item's ID correctly and it should display the options for an administrator to edit this shopping item, for instance, displaying the hyperlink to go into editing webpage of the product catalogs of this item, displaying the options if the administrator wants to edit the item's text description or delete it and so on.

Second, administrative module should display a webpage for an administrator to create product catalogs in selected shopping item and this administrative webpage should display a distinguished name, for example I defined it as 'Page Editor' in this module. In this page, it should display the ID of product catalogs and it should display the options for an administrator to edit the information of a product catalog, for instance, displaying the hyperlink to go into editing webpage of products of his catalog, displaying the option if the administrator wants to edit the catalog's description or delete it. Furthermore in this page, it should also display the options of editing the layout of product list of WAP shop.

Third, administrative module should display a webpage for an administrator to create products in selected product catalog and this administrative page should display a distinguished name as well, for example I also defined it as 'Page Editor' in this module. In this page, it should display the ID of a product and it should display the options for an administrator to edit the information of a product, for instance displaying an option of description methods including text description, graphic description and order price, displaying the option if the administrator wants to edit these descriptions or delete them. Furthermore in this webpage it should also display the options of editing the layout of the single product page.

Since the functionalities of uploading picture are specially required in administrative module, it should display an individual webpage for picture uploading.

All in all, the visualization functionalities of administrative module should correspond to the prototype of WAP shop defined in previous section. The layout of administrative module can be visualized as the following figure:

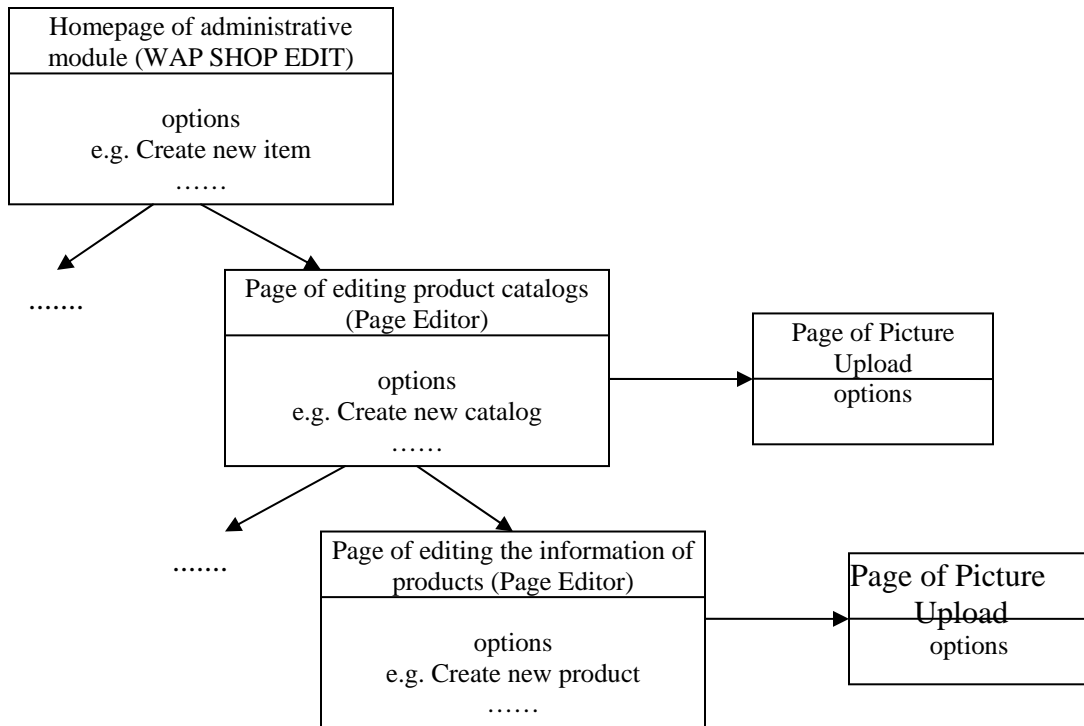


Figure 2.3 Prototype of Administrative module

Undoubtedly, since one item can consist of a number of different product catalogs, the number of webpages of editing product catalogs should be equal to the number of product catalogs, and this mechanism is similar to webpages of editing the information of products. The Figure 2.3 is only the general prototype of administrative module, which didn't list all webpages of editing product catalog and webpages of editing product information corresponding to Figure 2.2.

#### 2.3.4 The expectative management process (Editing functionalities)

A complete management process should begin from the administrator login, but in Section 1.3 Correlated work, I have already pointed out, the authorization functionality is not realized in WOSS system, so I omit this functional requirement in this thesis.

Administrative Module is to manipulate the contents displayed in the customer interface. First, it should provide the functionalities of modifying the layout of WAP shop, for instance, change the sequences of product catalogs, uploading the advertisement pictures of WAP shop and editing the text descriptions of WAP shop, etc.

Second, it should provide the functionalities of modifying the contents of items, for example the text descriptions of items and the displaying status of these items. (For instance, if the owner of the WAP shop does not want to display a specific item in

the special time, administrative module should provide this functionality, but don't need to delete this item).

Third, administrative module should provide the functionalities of modifying the contents of product catalogs, for instance, pointing the ID to a product catalog, pointing the displaying page ID to a product catalog, allowing the administrator to input the text description, allowing him to upload the advertisement picture, allow him to edit existing information and the status of displaying a specific product catalog that is same as the situation of items. In addition, it should also allow the administrator to change the layout of product catalog webpages of WAP shop.

Fourth, it should provide the functionalities of modifying the contents of products, for instance, pointing ID to a product, pointing the displaying page ID of a product, allowing the administrator to edit the text description of this product, allowing him to upload the picture of products, allowing him input or change the prices of products and allowing him to determine the status of a specific product (the same situation as items). Furthermore, it should provide the functionalities of changing the layout of product webpages of WAP shop.

These functionalities mentioned above can be defined as the editing functionalities required in WOSS system.

Now, I assume the administrator Mike is the user of Windows Internet Explorer, and he is going to use administrative module to edit the information of WAP shop. As the prototype of administrative module indicated above, Mike's management process should be the following steps:

1. Administrator login
2. Choose an item to edit, or create new items (Input description).
3. Click the item he wants to edit, link to another page editor.
4. Choose product's catalogs to edit, or create new product's catalogs (Input description)
5. Choose a specific product to edit, or create new products (Input description)
6. Choose the type to enter of edit, for example the type of text description, graphic description and price information.
7. Click on the buttons to add, edit, or delete
8. Finish product edit, back to step 3 to edit or add items, or back to step 4 to add or edit catalogs.
9. Repeat step 5-7 to edit or add new products.
10. Back to WAP SHOP EDIT homepage, log out.

### **2.3.5 Functional requirements of Database system**

Database system of Online Shopping System based on WAP will save all the data concerned with customer interface and administrative module.

First, it should record the data of customers, administrators, and system crew defined in Section 2.2 correctly. Second, it should record the customer's orders correctly. Third, the database system should also save the data of product catalogs and products correctly, for example, saving the advertisement pictures.

At the end, to make the database system to be connected to administrative module and customer interface correctly, it needs a proper configuration of database connector. In Chapter 4 and Chapter 5, I will give the detailed explanation of the database design and implementation.

## 2.4 Introduction of correlation between functionalities and system's files

From Section 2.1 to Section 2.3, I gave the descriptions of prototype of Online shopping system based on WAP in details. Now, the readers may understand how a WAP online shopping system runs, but I believe it is also necessary to give the very brief introduction of system design information to make a soft transition to the following chapters. This introduction is only focusing on the realization of functionalities, some system files must be mentioned and it can be defined as beforehand introduction of system structure. To get the detailed design information and system application files referred in this section, please consult Section 4.3, Section 4.4 and corresponding source from Appendix. In this introduction, I will use Customer Interface instead of WAP shop, or Customer Shopping Page employed above, and Administrative Module / Web server instead of Administrative interface. These substitutions are corresponding to the terms used in Chapter 4 System Design.

### Introduction of Customer Interface:

Functionalities	Corresponding System files	Realization Methods	Corresponding Sections (Design)	Corresponding Sections (Implementation)
Customer Login	wapShop.jsp	Servlet session	Section 4.4	Section 5.2.1
Display	extraView.jsp	Java.sql; WML, etc.	Section 4.4	Section 5.2.1
Customer Order	order.jsp; orderDel.jsp saveOrder.jsp	Servlet session	Section 4.4	Section 5.2.2

Table 2.1 Correlation of Customer Interface

### Introduction of Administrative Module:

Functionalities	Corresponding System files	Realization Methods	Corresponding Sections (Design)	Corresponding Sections (Implementation)
Administrator Login	WapShopImage.jsp	Servlet session	Section 4.3	Section 5.1
Create, Edit Items	WapShopImage.jsp; ewiupdate.jsp; ewidel.jsp	Java.sql; JavaScript form, etc.	Section 4.3	Section 5.1.1
Create, Edit Product Catalogs	PageEdit.jsp; ewidel.jsp	Java.sql; JavaScript form, tec.	Section 4.3	Section 5.1.2 Section 5.1.3

Create, Edit Products	PageEdit.jsp ewipageupdate.jsp	Java.sql; JavaScript form, etc.	Section4.3	Section 5.1.2 Section 5.1.3
Picture Upload	picupload.jsp	com.jspsamrt.upload.SmartUpload	Section 4.3	Section 5.1.2

Table 2.2 Correlation of Administrative Module

In Table 2.1 and Table 2.2, the realization methods are meaning the important methods or data structures used to realize the functionalities of WOSS system. For understanding the mechanism of Java.sql, readers can refer to Section 3.6.

Of course, in WOSS system development, there are more methods and programming languages than I stated above. For instance, the display methods of WML pages, HTML DOM, and JSP, etc. These tiny points of system development are introduced in Chapter 3, Development Tools, such as Section 3.2 WML, Section 3.4 Web Server, Section 3.5 JSP, and Section 3.7 JavaScript and HTML.

The correlations between Customer Interface and Administrative module are stated in Figure 4.4. Readers can refer to that figure. All these two parts connect to database through JDBC. Readers can refer to Section 3.6, and Section 5.1.4.

Undoubtedly, I cannot ignore the introduction of database management system. However, the description of database management system should go deeply into database development and database design that are discussed in Section 3.3 and Section 4.5.

## 2.5 Correlated work and conclusion

After a customer submits his order, WOSS system records this in database management system. The following tasks of a complete information management system should be management and processing of customer's orders (the tasks of so-called Financial Center), delivering customer orders to Logistic Center, distribution processes, and tracing the payment to Financial Center, etc. These include the implementations of ERP, CRM systems. But as in the beginning of this chapter indicated, those functionalities are not implemented in WOSS system. It needs the interface between every module, for instance the interface between WAP SHOP EDIT and Financial Center, the interface between Financial Center and Logistic Center, and dataflow into database system, etc. I leave these developments as future works which are not covered in this thesis.

All in all, in this chapter, I define the prototype of Online Shopping System based on WAP, and this prototype including the user classes, visualization functionalities and process functionalities both in customer interface and administration module and functional requirements of database system. These functionalities instruct the following selections of development tools, system design and system implementation. Of course, these required functionalities are fundamental and in Chapter 5 System Implementation I will give the detailed description of realized functionalities of WOSS system.

## 3 Development Tools

In this chapter, I introduce the development tools that are adopted to develop WOSS system, including the reasons why I chose these software, comparisons of different software and introduction of programming syntax, semantics briefly such as SQL, JSP, HTML and JavaScript, etc. These introductions of programming syntax are only corresponding to WOSS system functionalities. In other words, some parts of programming syntax that are not employed in WOSS system are not covered in this chapter.

The aim of giving these instruction of programming syntax is to provide readers a soft approach to the following chapters, and to make the reader easily understand the code quoted in the following chapters, system design and system implementation. In this chapter, all the codes quoted are in italic.

### 3.1 Introduction of development tools used in WOSS system

How to develop a complicated Online Shopping System based on WAP that provides the functionalities discussed in Chapter 2, what the proper development tools are for this EJB<sup>10</sup>-level system and to realize those functionalities are a complicated and cogitative research process.

In the beginning, I studied some existing online shopping systems, wireless e-commerce documentations, and consulted some of the experts who are working on the software development in wireless communications. In addition, I compared some software which provide various functionalities.

According to the prototype defined in Chapter 2, and Figure 2.1, I concluded the effective WOSS system should be divided into 3 parts, including customer interface, web server and database management system. In the customer interface development (in the theory of Software Engineering, we know it is defined as 'Presentation Layer'), I should adopt the development tools which are specific for designing WAP web pages and this interface can be browsed by PC users as well. In web server development (in the theory of Software Engineering, we know it is defined as 'Process Layer'), I should adopt the development tools which can assemble the Java applications, database connector, and any functionalities web server should provide. In database system development (in the theory of Software Engineering, we know it is defined as 'Data Layer'), I should adopt the development tool which is fast, and easy to connect to web server. Furthermore, since WOSS system is designed for academic research, the software that needs to pay should be avoided.

---

<sup>10</sup> EJB/J2EE: Enterprise JavaBeans, used at corporation level for large-scale web server programming, <http://java.sun.com/products/ejb/>, Sun Microsystems, 1994-2004

As the discussion above, the basic structure of WOSS system can be visualized as the following figure that indicates the functionalities and relations of these applications. Mobile clients use their cell phones that support WAP to access to WAP gateway, and PC users use their navigators to access WAP gateway. Therefore, customer's interface is developed by WML and HTML, and web server is developed by Orion application server, JSP, etc, which connects to MySQL database management system through JDBC.

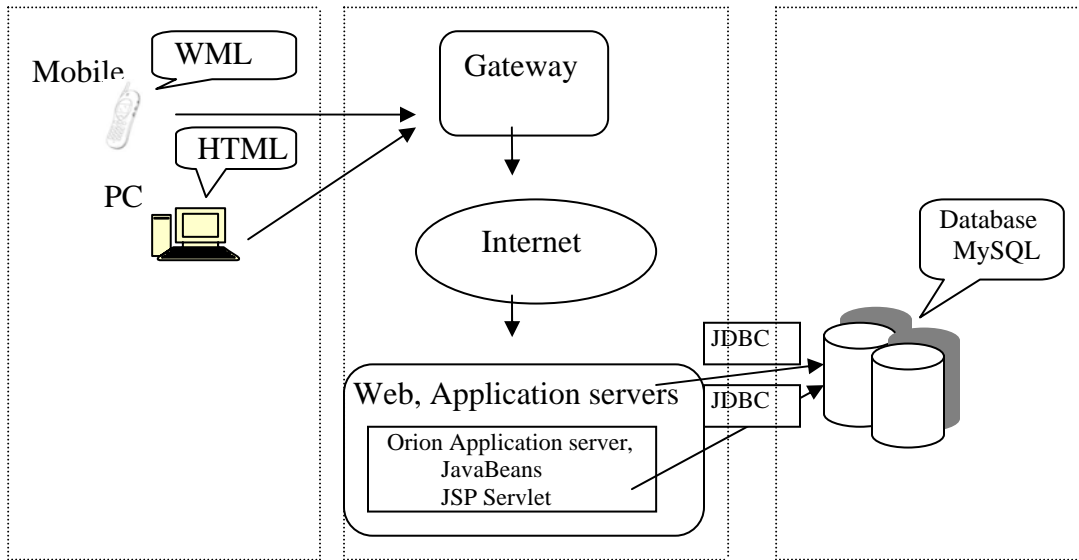


Figure 3.1 WOSS system structure

In web server development, both Orion application server and Tomcat are Java assemble products, but I chose Orion application server rather than Tomcat as the development tool of web server in WOSS system, not only because Orion application server supports EJB/J2EE standard, but also because it has the excellent feature in the compatibility with WAP applications. Although, Tomcat is developed by Apache organization and it has the superior supports from SUN Microsystem that develops J2EE, it still has the shortcoming of EJB applications. Microsoft Internet Explore is the default Internet navigator for administrative interface which is the internal web page only for WOSS system administrators to manipulate the contents shown to customers. I used JSP, JavaScript, and HTML to develop this administrative interface (WAP SHOP EDIT, Refer to Section1.1 key words) and it connects to database system through JDBC.

In database system development, I chose MySQL as database management system of WOSS system, not only because it's fastest and multi-threads, but also because it is open source software.

In customer interface development, I used WML and JSP to develop this part that can be browsed by Opera navigator and OpenWave navigator.



## 3.2 Using WML

Undoubtedly, HTML, to a software developer, is definitely familiar, because it is the fundamental programming language to develop Web pages. However, WML, as a specific mark-up language designed for WAP Web browsers, I guess not much many developers are proficient in this language. As a result, in this section, I introduce WML to the readers whereas very briefly.

WAP is the abbreviation of Wireless Application Protocol, which was found in 1997<sup>11</sup> by Ericsson, Motorola, Nokia, and Uwired Planet.

WML stands for Wireless markup Language, which is a mark-up language inherited from HTML. Since WML is based on XML, it is regulated much more strictly than HTML. For instance, **<card>** is not the same as **<CARD>**.

WML pages are displayed in a WAP browser, such as Opera, OpenWave, or cell phone browser which supports 2.5G standard, or higher than 2.5G. The pages in WML are called DECKS, which consists of a set of CARDS. Each card begins with **<card ...>** and ends with **</card>**, and different cards are related by links. When a WML page is accessed with WAP browser, all the cards in the pages are downloaded from the WAP server (see 1.2.1 WAP overview, WAP gateway and figure 1-1 WAP e-commerce), and only one card will be displayed at a time. But WML decks include several cards, in other words, a WML document or file can include a deck, and one deck can be embedded with several cards, which will be displayed in WAP browsers.

All the WML documents begin with:

```
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

Doctype is defined as WML, and DTD is accessed at [www.wapforum.org/DTD/wml\\_1.1.xml](http://www.wapforum.org/DTD/wml_1.1.xml). The WML content is inside the brackets, **<wml>.....</wml>** tags, and actual paragraphs are inside **<p>.....</p>**.

To define a formal WML document (embedded with JSP), a developer should do like this:

```
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<head>
<access/>
<meta ...../>
```

---

<sup>11</sup> WAP, [www.w3schools.com](http://www.w3schools.com), W3Schools, Refsnes Data, 1999-2004

```

</head>
<card>
<p>..... // contents in the paragraph.
</p>
<% ..... // JSP scriptlet tag, including valid Java code
%>
</card>
</wml>

```

**<head> ..... </head>** is adopted to define the related information of decks. For example:

```

<head>
<meta http-equiv="Cache-Control" content="max-age=0" />
// the deck will always be loaded off the server

<meta http-equiv="Cache-control" content="no-cache" />
//the directive CACHE-CONTROL defined in HTTP1.1: NO-CACHE indicates
//cached information should not be used and instead requests should be forwarded
//to the origin server
</head>

```

Several **<meta...>** which is to specify the meta information of WML file, **<access>** which defines the information about access control of WML files, are allowed inserted inside **<head>...</head>**.

Card elements define the properties and including contents of WML card. For example it can be defined as:

```

<card id= " " title= " " .....> // " " can be substituted by '

```

**id** is to define the card name, which is used to locate the card, for example by using **<go href="card name">** to navigate. (in WOSS system, most of the time I use **<go href="http://localhost/xxx.jsp" />**)

**title** is to present the title of WML page and descriptions, of course it can be combined with SQL query, for instance in **extraview.jsp** (Appendix 8.2.8):

```

<card id='welcome' title=' '+rs.getString("text").trim()+" '>

```

I use **getString** method of **ResultSet** to retrieve the text information and give it to card title.

The syntax and semantics of WML are quite similar as HTML, but stricter. I give the description of typical WML tags which are used in WOSS system.

<br/> defines a line break.  
<p> ... </p> defines a paragraph.  
<table...> ... </table> defines a table.  
<td> ... </td> defines a table cell.  
<tr> ... </tr> defines a table row.  
<do type="type"> ... task... </do> defines an activation of a task when cursor of user's mouse clicks on a word or phrase on screen.  
<go href= "url"> ... </go> defines a switching to another card.  
<prev>... </prev> defines a returning to previous visited card.  
<b>...</b> defines a bold sentence.  
<i>... </i> defines an italic sentence.

### 3.3 Using MySQL

Microsoft Access, SQL server, and Oracle, etc are quite familiar to the readers. There is another database management system which might be heard by readers at the first time. MySQL is the database management system chosen in WOSS system development. In this section, I give the description of MySQL and some important configurations that were recorded in the process of development are also quoted from WOSS system development, which can help readers to well understand the mechanism and implementation of MySQL database.

In WOSS system, I chose MySQL version 1.4 (for Windows NT) as the database tool. MySQL is the most popular SQL database management system in the global market, not only because it delivers a very fast, multi-threaded, and multi-users database services<sup>12</sup>, but also because developer can choose MySQL as an open source database tool , in other word, it's free to download and use by non-commercial users.

In WOSS system, MySQL is a database management system which is located in Data Layer, and employed for saving all kinds of information of WapShop (Shopping facilities, reference to 1.1 Key words, WapShop) and WAP SHOP EDIT (Administrative facilities, reference to 1.1 Key words, WAP SHOP EDIT), such as user information, products information, and order information, etc.

MySQL stands for SQL (Structured Query Language) which is most common, and standardized database programming language. MySQL's SQL is defined by the ANS/ISO. In this thesis, the current version of standard, SQL 2003 has been referred

MySQL database system can be downloaded from [dev.mysql.com/tech-resources/articles/4.1/installer.html](http://dev.mysql.com/tech-resources/articles/4.1/installer.html), as a typical Windows installer packet. MySQL should be installed just under the C:\ directory, with the default folder name 'mysql'.

WOSS system is developed in Windows system, in other words, MySQL database server is installed in Microsoft OS. To using MySQL services, first of all, a user

---

<sup>12</sup> MySQL reference manual version A4, MySQL AB, 1997-2004

should have the authorization of Windows administrator, and launch the command line, by using general DOS commands. Second, enter the directory C:\mysql\bin, by typing 'net start mysql', a user can start MySQL services at the local machine, and since MySQL employs TCP/IP protocol to connect client machine to MySQL server, local machine should install the general Windows TCP/IP protocol.

Since MySQL database is employed for save all kinds information of WOSS, it has critical authorization mechanism. The default username and password of MySQL database server are 'root', which is saved in default database 'mysql' of the whole database server, and without the prearranged password, in other words, anyone who has Windows administrator account can enter MySQL database server.

By typing 'mysql -u root' after c:\mysql\bin, a user can enter mysql server. -u stands for 'user'; 'root' is default username of MySQL, and without password.

By typing > mysqladmin -u root -password, a user can create a password for default root, and by typing > mysqladmin -u root -p *old\_password* password *new\_password*, a user can modify the password of root. For instance:

*mysql > mysqladmin -u root -password 123* , which sets password 123 for root  
*mysql > mysqladmin -u root -p 123 password 456*, which change the password, 123 to 456 for root.

By using username 'root', with the password '456', the user has all the default privileges for root, including:

privileges	rows	domains
select	Select_priv	Tables
insert	Insert_priv	Tables
update	Update_priv	Tables
delete	Delete_priv	Tables
index	Index_priv	Tables
alter	Alter_priv	Tables
create	Create_priv	Databases, tables, indexes
drop	Drop_priv	Databases, tables
grant	Brant_priv	Databases, tables
references	References_priv	Databases, tables
reload	Reload_priv	Database management
shutdown	Showdown_priv	Database management
process	Process_priv	Database management
file	File_priv	Database(files)management

Table 3.1 MySQL privileges of 'root'

Under the username 'root', administrator can test MySQL server, to check if it is running in gear or not. For instance:

```

>mysql -u root -p 456
>show databases;
> +-----+
| Databases |
+-----+
| mysql     |
+-----+

```

mysql database is the default database of MySQL server. By typing 'show tables'; administrator can check the tables of MySQL database.

```

>mysql -u root -p 456
>use mysql
>show tables;
> +-----+
| Tables   |
+-----+
| columns_priv |
| db        |
| func     |
| host     |
| tables_priv |
| user     |
+-----+

```

MySQL database has six default tables. User table save the user's privileges, such as username and password. 'Columns\_priv' and 'tables\_priv' grant tables and columns specific privileges. 'db' and 'host' tables grant databases specific privileges. 'func' save the name of function, type and sharing database name. For example, when the users request shutdown, reload, insert, or update to MySQL server, the server will firstly check user table and identify users. Host table saves the list of servers, including all the authentic PCs on the local net.

Under the username 'root', administrator can install new users in MySQL database, by using 'insert', for instance:

```

>mysql -u root -p 456
>use mysql; (which means select the default database of MySQL server.)
>insert into user (host, user, password) values ('localhost', 'Mike', password
('123'));

```

which means I installed Mike in localhost with the password'123'.By using 'grant', administrator can set the privileges for users, for instance:

```

>mysql -u root -p 456
>grant all privileges on *.* to Mike identified by '123' with grant option;

```

which means I set '123' as the password of Mike and installed this username into server.

In MySQL, there are two table types which can be chosen, namely, MyISM and HEAP. In WOSS system, I only employed MyISM which is default table type of MySQL. For example, a user can update a new row of table without deleting previous contents, and it supports *varchar* data type.

MySQL supports the interaction between Java programming and database by providing Java Database Connectivity (JDBC) driver. It helps a programmer to build a Java programming which connects to database data easily. I downloaded the latest JDBC driver from MySQL official website.<sup>13</sup> The technical details of Java programming connecting to MySQL database through JDBC will be presented at Chapter 4.

MySQL database server has a suit of complicated authorization and privileges system. And this mechanism guarantees the functionality of connectivity between remote access PC and local host, and convergence between users and the server. In the real WAP e-commerce system, MySQL authorization system is tightly related to interface encrypt mechanism. Since this functionality has no direct relationship with realization of the prototype defined in Chapter 2, and it could be as complicated as the development process of WOSS system, I will not present more detailed and future researches on MySQL authorization system. From the introduction above, readers may understand how I developed or operated the database management system in WOSS system.

### **3.4 Using Orion Application server**

Web server is the key subsystem of the whole WOSS system. Consequently, to choose a proper applicable software is one of the important tasks in developing WOSS system to me. In this section, firstly I describe one of the most popular J2EE technologies, JavaBeans, and secondly I introduce Orion Application Server to readers.

#### **3.4.1 JavaBeans overview.**

Based on JavaBeans Application specification<sup>14</sup>, JavaBeans technology is the component architecture for the Java 2 Platform, Standard Edition (J2SE). Components (JavaBeans) are reusable software programs that programmer can develop and assemble easily to create sophisticated applications.

---

<sup>13</sup> JDBC driver of MySQL: [http://dev.mysql.com/doc/mysql/en/Java\\_Connector.html](http://dev.mysql.com/doc/mysql/en/Java_Connector.html), MySQL AB, 1997-2004

<sup>14</sup>JavaBeans Application specification: <http://java.sun.com/products/javabeans/>, SUN Microsystem, 1994-2004

As I have written in the Chapter 1, SUN Microsystem has assemble a varieties of Java technologies to create a Java web server, JavaBeans, and now it's more or less like a standard for these kinds of technology assembles, for example, Tomcat, Orion application server, etc. can also facility a Java programmer to develop applications at so-called Java assemble container which is more less like a vessel containing Java applications, and some of programming juncture applications are existing. Programmers may just save the applications, for example, JSP files into the default directories of JavaBeans application server, and start the JavaBeans services in command line. These act as a full-features Java web server instead of developing every web application by programmers.

### 3.4.2 Orion Application server

Orion Application Server is a complete and excellent application of JavaBeans Technology. I downloaded the package for Windows OS of Orion application server from Orion application official site<sup>15</sup>. Orion application server package is granted to the licensee a non-exclusive, non-transferable and non-assignable right to use the software on one license, and it's not a shareware for commercial development which means firms develop software applications for profits, except academic or non-profits development and research.

A general Orion application server sketch can be visualized by UML Component and deployment diagrams as follow:

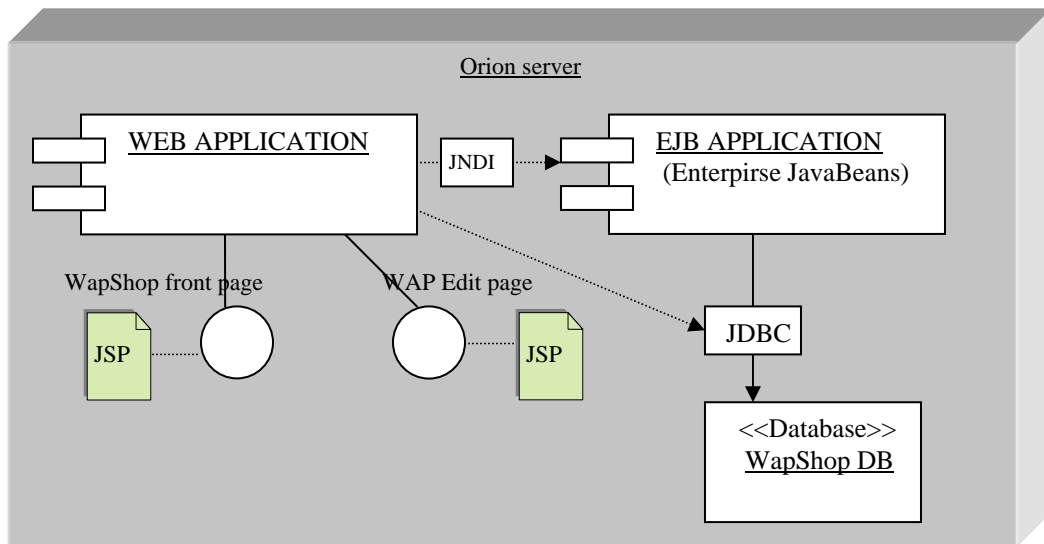


Figure 3.2 Sketch of the Orion application server

JNDI stands for Java Naming and Directory Interface<sup>16</sup>

<sup>15</sup> Orion application official site: [www.orionserver.com](http://www.orionserver.com) IronFlere

<sup>16</sup> JNDI: <http://java.sun.com/products/jndi/>, SUN Microsystem, 1994-2004

EJB application stands for enterprises JavaBeans application

Web application including several web pages such as WAP pages developed by JSP and WML, is connected to EJB application by JNDI, and EJB application communicate the data with database system through JDBC. This is a general sketch of Orion application server. Actually, in development of WOSS system, since there are only two kinds of customers, WAP mobile customers and PC users owning WAP connection, connect to Web server, JNDI applications are not applied and it is not necessary to configure JNDI applications for a small scale of web application usage. Furthermore, since EJB is designed for a large scale of JavaBeans server application, I only employed part of EJB functionalities in development of WOSS system. The rest of applications and functionalities shown in Figure 3.2 are fully employed in WOSS system.

Installation of Orion application server is quite easy.

First, a user should unzip the package download to C:\, named a directory as orion. Second, for enabling JSP or other applications which require to access to Java compiler, a user should copy the file, tools.jar which is located in the JDK\lib root from J2SK1.4.2\_04, or other version installation to created orion directory. In development of WOSS system, I have done as follows:

*copy c:\jdk1.4.2\_04\lib\tools.jar c:\orion200410082037*, which might be different copy commands and directory installed java runtime environment.

Orion application server was installed in my Windows XP professional C:\. And it is using port 80 for web server part. In other words, for the developer who wants to run the web server at local host, should enable port 80. And if port 80 has already been occupied for another web server, such as Tomcat, developer should change this configuration. In development of WOSS system, I have done as follows:

Enter the file config/default-web-site.xml, change *<web-site display-name="Default Orion WebSite">* to *<web-site port="8080" display-name="Default Orion WebSite">*

So far, the installation of Orion application server at local host is completed. All the JSP files deployed are saved at the directory **C:\xxx\Orion\default-web-app\ExtraWapImage**. ('xxx' stands for a discretionary directory name)

Datasource connect loader, for example JDBC, is saved at the directory **C:\xxx\Orion\default-web-app\WEB-INF\classes\com\gtom\wap**.

Uploaded data in web server, for example the uploaded pictures are saved at the directory **C:\xxx\orion\default-web-app\wap\images\uploadpic**.



'xxx' stands for the folder named by developers as their favorites. To start the services of Orion application server, I enter the directory by typing:

```
>c:\cd orion200410082037\orion  
>c:\ orion200410082037\orion\c:\j2sdk1.4.2_04\bin\java -jar orion.jar
```

Afterwards, if the service has already been successfully started, 'Orion/2.0.2 initialized' should be appeared on the screen. And open the Internet Explorer which is designed as the default browser of WAP SHOP EDIT, by typing `http://localhost` into URL address, 'Orion Application Server 2.0.2 initialized' appears.

To make readers easily understand the mechanism of Orion application server, I introduce some important component tools of Orion application server which are adopted to realize some important functionalities of web server of WOSS system. They are:

The main server tool: **orion.jar**, which is used for installing the server, activates the admin account and rewrites text files to match the OS linefeed, compressing standard output, enabling context lookup support from user-created threads, launching the management console in process, reporting the error output, etc.

The graphical tool for assembling J2EE applications: **earassembler.jar**, which deploys for optional path to J2EE application.

Orion management console: **orionconsole.jar**

The graphical tool for assembling tag extension libraries: **taglibassembler.jar** which deploys optional path to taglib, such as JSP (taglib).

The graphical tool for assembling Web applications: **webappassembler.jar**, which deploys optional paths web applications.

Load balancer and clustering application: **loadbalancer.jar**, which deploys HTTP connections, such as listen the host port.

Administration tool for Orion application server: **admin.jar**, which deploys to configure of username, password; shutdown the server; install the global applications; restart the server and add name users, JDBC url, etc.

DataSource<sup>17</sup> was introduced in JDBC, as an easier way of obtaining a JDBC connection and shielding the developer from issues regarding configuration such as database connection pooling. Orion application server provides a Datasource wrapper

---

<sup>17</sup> DataSource: [www.orionserver.com](http://www.orionserver.com), , IronFlere

which works as an emulator for JDBC connection drivers and optional connection pooling. It is named as JDBC connection driver for MySQL database system: **mysql-connector-java-3.0.9-stable-bin.jar**.

Uploading files for web server application: **jpsmartupload.jar**, which deploys to realize the functionality of uploading files.

### **3.5 Using JSP**

JSP is the universal programming language used in developing WOSS system, not only all the application files of Web server (WAP SHOP EDIT), but also the Customer Interface. In this section, I give the brief introduction of JSP mechanism and syntax as the prerequisite information of system design and system implementation. These also help the readers to understand the code files attached in appendix. Some source code will be quoted from WOSS system for readers well understanding the application JSP files listed in appendix.

#### **3.5.1 JSP overview**

JSP stands for JavaServer pages which is developed by SUN Microsystem. It is a technology based on Java language and enables the development of dynamic websites. JSP enables server side development, and it mixes Java code, HTML with special tags, and other mark-up languages, such as WML to provide dynamic contents.

As a rule, JSP source code runs at the web server in a JSP servlet engine, which dynamically generates HTML, WML, etc. and send them output to the client's web browsers. First, developer creates a JSP file named its extension as '.jsp', including Java code and HTML with special JSP tags. The JSP engine parses the .jsp file and creates a Java servlet source file which means JSP servlet generates a special servlet from JSP file and all the HTML, or WML required are converted to println statements. It then compiles the source files into a class files, and all of these processes are done at the first time. Afterwards, the servlet is instantiated. HTML or WML from the servlet output are sent via the Internet, and displayed on the end users' screens. The architecture of JSP can be visualized as Figure 3.3. The fundamental development tools of JSP of WOSS system are text editor J2SDK, and web server, such as Orion application server, Tomcat, or any J2EE reference implementation.

#### **3.5.2 JSP configuration**

I download JDK and J2EE for Windows OS from the official SUN webpage: <http://java.sun.com/j2se/1.4.2/>. One of the significant problems of configuration Java environment is to set the Java Path and ClassPath. In Windows OS, the Java codes should be saved at c:\j2sdk1.4.2\_04\bin, and if developer wants to compile these Java

files, he or she should enter the command line and under that directory to run ‘javac, java, javah, etc’. It is quite redundant to type this entire path every time developer compiles the java files.

As result, I change configure the Java path and ClassPath in autoexec.bat file, which is located at C:\. Since c:\j2sdk1.4.2\_04 is the installation path of Java program, so I wrote:

```
‘SET PATH=%path%; C:\j2sdk1.4.2_04\bin;’
```

Since dt.jar and tool.jar are the classes referred by Java editor, which are located at c:\j2sdk1.4.2\_04\lib, I wrote:

```
‘SET CLASSPATH=.; C:\j2sdk1.4.1\lib\tools.jar;
C:\j2sdk1.4.1\lib\htmlconverter.jar;C:\j2sdk1.4.1\lib\dt.jar;’ into Autoexec.bat file.
```

As I have introduced in Section 3.4.2 Orion application server, for Orion web server and its free source JSP and servlet engine, I have set the default port of local PC for web browser is 80. In other words, any applications which occupy this port should have changed the port value.

The architecture is shown as follows:

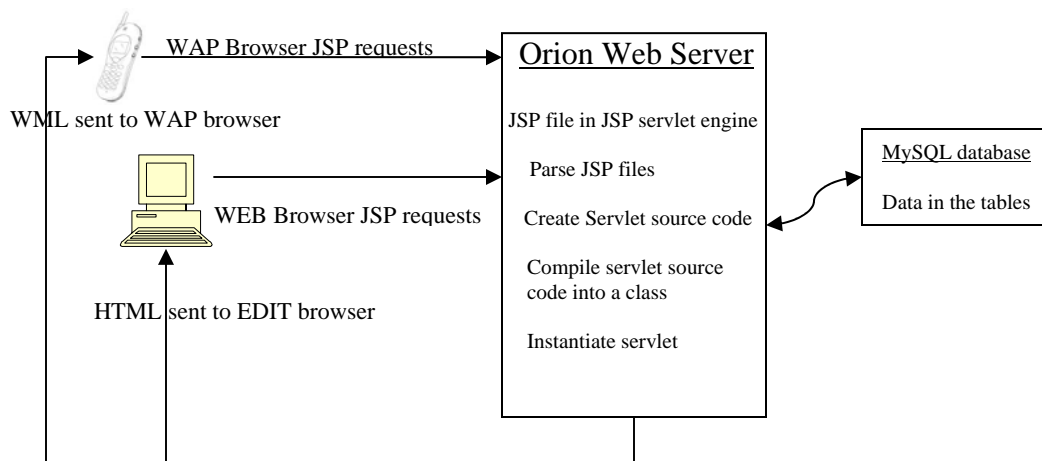


Figure 3.3 JSP Architecture

### 3.5.3 JSP syntax and simple examples of WOSS

JSP programming language includes java code, HTML and special JSP tags. In the beginning of development of WOSS system, I have written such a simple JSP file to test my Orion application server and its JSP servlet engine:

```
<html>
<head>
<title> the test page for Orion application server configuration
```

```

</title> //ending title in HTML
</head> // ending head in HTML
<body>
<%@ page language = "java" %>
// directive tag, to describe which language the file uses
<% system.out.println ("the test page of Orion application server");%>
// scriptlet tag, the simple java code for print a sentence.
</body>
</html>

```

I saved this file into \orion\default-web-app, started the service of Orion application server, and called it via my web browser, the sentence ‘the test page of Orion application server’ appears on my web browser.

In the code above, I have referred the directive tag, scriptlet tag, and there are still some other main tags in JSP language of WOSS system. They are: action tag, declaration tag, expression tag, HTML comment tag, Hiding comment tag, and so on.

**Declaration tag:** <%! .... %>, which is the usage of the declaration of variables and methods. For instance:

```

<%! int i=0; // defines ‘i’ as integer and from 0.
String foo= "Hello";
private void bar() {... }
%>

```

**Action tag:** <jsp: .... />, which is the usages of enabling server side JavaBeans, calling one JSP page from another, invoking an applet on the client browser and browser independent support for applets. For instance in picupload.jsp, (Appendix 8.2.6)

```

<jsp:useBean id="myUpload" scope="page"
Class="com.jspsmart.upload.SmartUpload" />

```

// calling JavaBeans which includes picture upload method and id is myUpload, with class name com.jspsmart.upload.SmartUpload.

There are four types of JavaBeans scopes.

Scope="page" means valid until page completes.

Scope="request" means bean instance lasts for the client request.

Scope="session" means bean lasts for the client session.

Scope="application" means bean instance created and lasts until application ends.

**HTML comment tag:** `<! .... >`, which is the usage of generating a comment and sending to clients. For instance in wapShop.jsp (Appendix 8.2.7):

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

**Directive tag:** `<%@....%>`, which is usage of description of special information about this JSP file to the JSP engine. For instance in ExtraWapImage.jsp (Appendix 8.2.1):

```
<%@ page language="java"%> // declaration of Java language being used
<%@ page import="java.sql.*"%> // load the JDBC driver
<%@ page import="com.gtom.wap.*"%>
<%@ page session="true"%>
// the value of session is true, and the session object refers to the current or new
session.
```

There are several different attributes for page description<sup>18</sup>:

Language: which language the JSP file uses.

Extends: Superclass used by JSP engine for the translated Servlet.

Session: does the page make use of sessions, and whether the client must join an HTTP session in order to use the JSP page

Buffer: control the usage of buffered output for JSP. In WOSS system, I use 8kb.

Info: to provide the information and document for JSP page. For instance: developer name and version, etc.

**Scriptlet tag:** `<%....%>`, which is the usage of saving valid java, JavaScript code. For instance, in wapShop.jsp (Appendix 8.2.7)

```
<%session.setAttribute("name","Mike"); // session attribute to Mike
    session.setAttribute("phone","13641366774");
    response.sendRedirect("extraview.jsp");
// redirect current page to extraview.jsp.
%>
```

**Hiding comment tag:** `<%-- .... --%>`, which is usage of giving comment string or reference in JSP file, and it's not output to client.

### 3.6 Using JDBC

JDBC, as the communication connector between MySQL database and Web server, is very important to be configured. In this section, I give the detailed instructions of

---

<sup>18</sup> JSP attributes for page description: <http://java.sun.com/products/jsp/docs.html>, SUN Microsystem, 1994-2002

JDBC configuration and corresponding SQL syntax, for example the regular way to construct JDBC connection and the methods of data retrieving. These instructions are very important for readers to understand the source code and system design part. It will be realized in WOSS system in Chapter 5 System Implementation. Now I begin to introduce the basic concept of JDBC and the steps to use JDBC.

JDBC offers a standard library for accessing relational database. The primary JDBC objects represent connections to a database and the statements performed using those connections. The two basic kinds of statements used with a relational database are *queries* and *updates*.

The meaning of a standard library which is JDBC API is exactly abided the Java syntax. It doesn't attempt to standardize the SQL syntax, although JDBC standardize the mechanism for connecting to database, and the syntax for sending queries and updates, even the acquired data structure. So the developers can use general SQL syntax to construct their database system and JDBC allows them to change the database hosts, ports by declaring at specific position and code, such as

`jdbc:mysql://[hostname][:port]/dbname[?param1=value1][&param2=value2]...`

J2SE API defines the standard Java.sql package specially used in JDBC of WOSS system including

- interfaces**, such as java.sql.resultset, java.sql.connection, java.sql.statement;
- classes**, such as java.sql.drivermanager, java.sql.time;
- exceptions**, such as java.sql.SQLException.

The hierarchy of java.sql package can be shown as the following tree, and the methods referred to the usages of context's example.

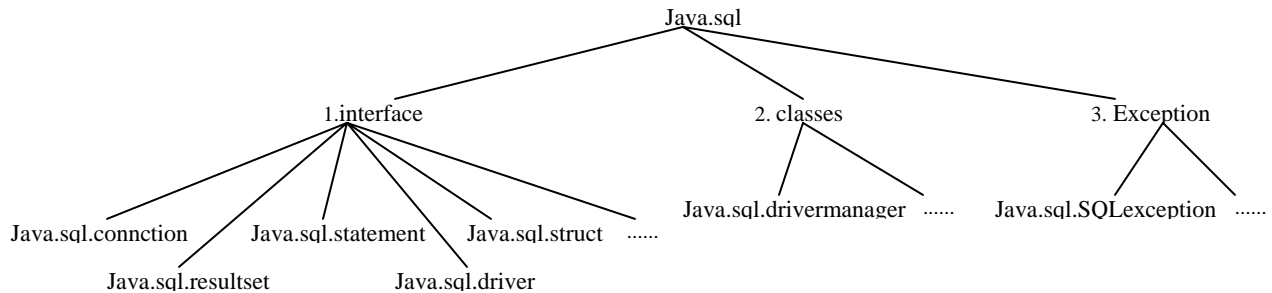


Figure 3.4 Java.sql tree

The methods included in java.sql.xxx can be shown as following tables:

objects	methods
Java.sql.conncion	createStatement(), close(), etc.
Java.sql.resultset	getObject(), getString(), getInt(), getMetadata(), close() etc.
Java.sql.statement	execute(), executeQuery(),etc.
Java.sql.driver	connect(),etc.
Java.sql.struct	getAttribute(), etc.
Java.sql.drivermanager	getConnection(), etc.
Java.sql.SQLException	getErroCode(), SQLException(), getNextException(),etc.

Table 3.2 Java.sql Methods

The fundamental steps of using JDBC connection has been standardized by lots of JDBC manual books and documentations. Now I just elicited the several important steps corresponding to WOSS system developing.

**The first step** is to load the driver which makes the communication with databases. I loaded a class with a static block. It makes a driver instance and registers<sup>19</sup> it with Driver manager, by using Class.forName. This method takes a string representing a fully qualified class name, and it could throw a ClassNotFoundException, so try/catch blocks are inevitable. For instance:

```
try{
    Class.forName(".....");}
catch (Exception E){
    System.err.println("Unable to load driver.");
    E.printStackTrace(); }
```

**The second step** is to specify the location of database server, which can be named as 'define the connection URL', and establish the connection, by using getConnection defined in java.sql.DriverManager, for instance:

```
connection conn = DriverManager.getConnection("jdbc: Database URL...");
```

**The third step** is to create a statement. It executes a static SQL statement that is the object adopted to send queries and commands to database, and returns results it produces. A statement can be created by:

```
Statement stat = Conn.createStatement() // it's defined in java.sql.connection
```

And by using execute method to execute the given SQL statement. For instance: To execute a query, by using executeQuery method that returns ResultSet object:

```
String sql=null;
```

---

<sup>19</sup> Practical statistics, 10700 times of 'load the driver', 98200 times of 'register the driver' in www.google.com, so I would like to use 'register the driver' in my thesis. Google Searching, 2004

```

ResultSet rs=null;
.....
if(pages==0)
    sql="select * from ExtraWapImage where type='top' order by orderid desc";
else sql="select * from ExtraWapImage where pageid="+pages+" order by
orderid desc";
rs=stat.executeQuery( sql );    //rs stands for 'ResultSet', stat stands for
'Statement'

```

There are several other methods of statement object, such as clear (), setMaxRows(), setMaxFieldSize(), etc. please consult J2SE API documentation<sup>20</sup>.

**The fourth step** is to process the results, by using ResultSet.next method. It is to process the table a row at a time, and it provides different getXXX method<sup>21</sup>, (since there are hundreds getXXX methods defined by Java.sql.ResultSet documentation, I only give the important examples used in WOSS system) such as:

**getString()**, which retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java language;  
**getInt()**, which retrieves the value of the designated column in the current row of this ResultSet object as an int in the Java language;  
**getObject()**, which acquires the value of the designated column in the current row of this ResultSet object as an Object in the Java language.  
**getMetaData()**, which Retrieves the number, types and properties of this ResultSet object's columns, etc.

Another important method of java.sql.ResultSet is **close()**, which releases the ResultSet object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.

For instance:

```

<% while (rs.next ( )) {%> // rs stands for ResultSet.
<tr>
<td align="center"><%=rs.getString("id")%></td>
// getString stands for displaying results regardless the column type.
// displays the retrieving 'id' in the table.
.....
</tr>
.....
<%
if(rs.getInt("status") == 1){
out.print("value='on'");

```

<sup>20</sup> J2SE API documentation: <http://java.sun.com/j2se/1.4.2/docs/api/>, SUN Microsystem, 1994-2004

<sup>21</sup> getXXX methods: Methods inherited from interface java.sql.ResultSet, [java.sun.com/j2se/1.4.2/docs/api/](http://java.sun.com/j2se/1.4.2/docs/api/). SUN Microsystem, 1994-2004.



```
// displays value='on' on the web browser
out.print("checked"); }
%>
<% if(rs!=null) rs.close(); %>
```

**The fifth step** is to close the connection, by using close () method (different from java.sql.ResultSet.close() which is to free the resultset object's database) defined in java.sql.connection. It is to release the connection object's database and JDBC resources immediately instead of waiting for them to be automatically released. For instance:

```
if(rs!=null)rs.close();
conn.freeConnection();
Conn.close();
```

### 3.7 Using JavaScript and HTML

JavaScript and HTML are definitely not strange for a sophisticated Web developer. But some parts of its syntax and semantics might be not familiar to specific readers, so in this section I give the description of major JavaScript and HTML syntax used in WOSS system. Similarly, some source code will be quoted from WOSS system for readers well understanding the application JSP files listed in appendix.

JavaScript<sup>22</sup> is a compact, object-based scripting language for programming client and server Internet applications. Most of the Internet navigators presently are all supporting JavaScript. For example, in WOSS system, Internet Explorer, as my default administrative Internet navigator, interprets JavaScript statements embedded in the HTML page. In other words, Client-side JavaScript is an interpreted language which doesn't need preliminary compilation.

Client-side JavaScript statements embedded in an HTML page can respond to user events, for instance, mouse clicks, form input, or dropdown list and page navigation.

Since JavaScript embedded in HTML is one of the most popular scripting languages, and its syntax and semantics are relatively easier than Java or C++, I only give some examples which reflect WOSS system's functionalities.

#### 3.7.1 HTML DOM event

When a user moves the mouse to click on an object, it appears an alert box with some contents, by using DOM event. For instance:

---

<sup>22</sup> JavaScript Tutorial: [www.w3schools.com](http://www.w3schools.com), Refsnes Data, 1999-2004

```

// when mouse's cursor click on 'alert.gif' picture located at Explorer, 'this is an
example' appears.
```

### 3.7.2 Create a table

To design a table which has two rows and three columns as below:

100	200	300
400	500	600

Developer should input the HTML code as:

```
<table border="1">
<tr>
  <td height="27" align="center">100</td>
  <td height="27" align="center">200</td>
  <td height="27" align="center">300</td>
</tr>
<tr>
  <td height="27" align="center"> 400</td>
  <td height="27" align="center"> 500</td>
  <td height="27" align="center">600</td>
</tr>
</table>
```

### 3.7.3 DOM checkbox

The Checkbox object represents a checkbox in HTML form. It is created by:

```
<input type="checkbox" value="on">
// 'value' Sets or returns the value of the value attribute of the checkbox
// 'onClick' is an event that executes some code when the checkbox is clicked
```

### 3.7.4 HTML input form

A HTML form is an area that contains form elements. For example, users can input data or information into a form. It is created by:

```
<form>
Input your ID
```

```
<input type="text" name="ID">
<br>
Input your Name
<input type="text" name="full name">
</form>
```

### 3.7.5 Dropdown list in a form

Dropdown list is convenient for users to make options. It is created by:

```
<select name="postion" onchange="put()">
  <option>center</option>
  <option>left</option>
  <option>right</option>
// create a dropdown list with position options, center, left, and right.
```

The basic HTML and JavaScript syntax has been stated as above. The detailed information about implement WAP SHOP EDIT page is presented at Chapter 5 System implementation.

### 3.8 Selecting Operating System

In fact, to make a choice of operating system which is employed to develop the WOSS system is not a critical problem. Since all the software I chose to program support Windows OS, and it has an excellent GUI interface, I decided to use it as my programming platform.

But the most Open Source software are designed in UNIX, Linux systems, and mass of sophisticated programmers choose these two as their programming platform. Consequently, WOSS system can also be designed at UNIX and Linux systems. The major difference is only the software versions which are compatible to UNIX, and Linux, installation, and Database driver.

### 3.9 Conclusion

In this Chapter, I gave the description of system development tools, including, for instance, the characteristics of development tools, their advantages, and comparison with others. I also gave the general introduction of programming syntax and semantics, such as SQL syntax. From these, readers can fundamentally understand the reason why I chose these tools, and how I adopted these tools to realize the prototype defined in Chapter 2. In next chapter, I will give the explanation of system design that includes customer interface design, administrative module design and analysis and design of MySQL database system.

## 4 System Design

In this chapter, I present the designing information of WOSS system. First, I give the description of system structure that includes three main modules of WOSS system approached by system architecture and their functionalities. Consequently, I give the designing plan for corresponding modules.

Second, according to the prototype defined in Chapter 2, and the discussion of development tools in Chapter 3, I present the arrangement of system files, for example, the plots of JSP files and their functionalities. Furthermore, the correlations between these JSP files will be presented by a figure.

In the end, I present the design information of MySQL database management system by using E/R graphic analysis and the details of table fields are also described in this chapter.

### 4.1 Overview

According to the prototype defined in Chapter 2, Figure 2.1 indicated Online Shopping System based on WAP should include three main parts, Customer Interface, Administrative Module and Database management system. As a result, I should follow this prototype to design WOSS system. In Chapter 3, Section 3.2 I have already presented the draft structure of WOSS system and in this structure, different modules are responsible for different functionalities described in Section 2.3. And in single module, there should be more than one application file which is in responsible for a specific task. So I designed five specific JSP files which serve Customer Interface module to realize the visualization functionalities and ordering functionalities defined in Section 2.3.1 and Section 2.3.2. In addition, I designed seven specific JSP files which serve Administrative module to realize the visualization functionalities and editing functionalities defined in Section 2.3.3 and Section 2.3.4. In the end, one Java file was designed to realize the communication between Orion application server and MySQL database system. The functionalities and correlations between them are introduced in Table 4.1, Table 4.2 and Figure 4.4. All the applicable JSP files are saved in orion\default-web-app which was introduced in Section 3.4.2, and JDBC java file is saved at default-web-app\WEB-INF\classes\com\gtom\wap.

### 4.2 WOSS system structure

To realize the prototype defined in Figure 2.1, WOSS system is designed separately with three major modules, which can be visualized as Figure 4.3.

User interface module is to simulate the customer login, present the online shopping web pages to customers and submit customer's shopping requests. This module is designed for both mobile users and PC users. And designing information of this module can be shown as:

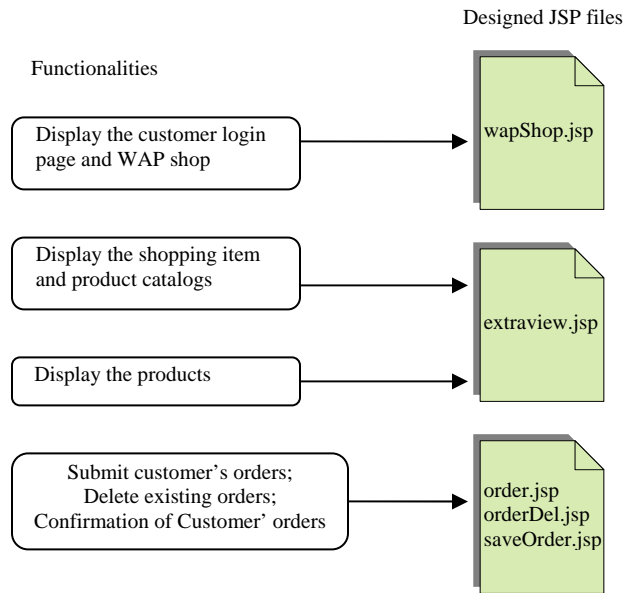


Figure 4.1 The arrangement of JSP files in Customer Interface

Administrative module is to control the web server. It is designed to create, add, and edit shopping items; create, add and edit product catalogs; create, add, and edit product information; modify the layout of shopping pages and administrative interface; and send the data corresponding to customers and WAP shop to database through the database connector. The designing information of this module can be shown as:

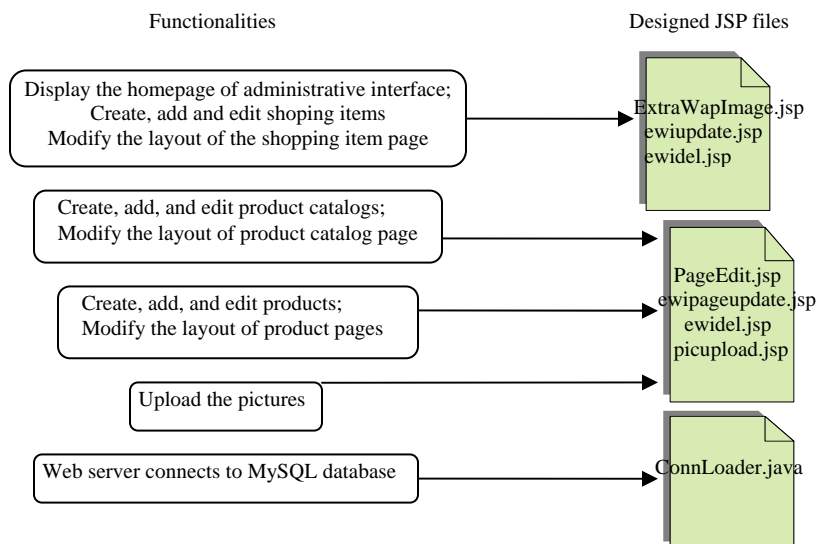


Figure 4.2 The arrangement of JSP files in Administrative Module

Database module is to save the product information, customer information, customer orders and any data related to WOSS system. For instance, the product tables save product's title, price, and picture, etc; the userorder table save users submitted order which distinguished by user's telephone number. In development of WOSS system, I designed 5 tables in MySQL database system and named this database as gtom\_wap. Every table has its distinguishing name, and the detailed designing information will come out in Section 4.5.

The main functionalities of these major three modules have been realized in WOSS system, and they are all instructionally designed by the prototype defined in Chapter 2. Some other extra functions like user authorization, customer feedback services are only left for the future work.

For getting a visible structure of WOSS, the following figure is presented. An administrator manipulates the administrative interface that designed in JSP files to modify any information concerned with WAP shop. And web administrative module sends this information to customer interface through Internet and WAP gateway. Customers browse acquired information through their WAP navigators and feedback the data to web server and save these data in MySQL database system through the components provided by Orion application server and designed JDBC connector. In MySQL database, all the data related to WAP shop are saved in the distinct tables.

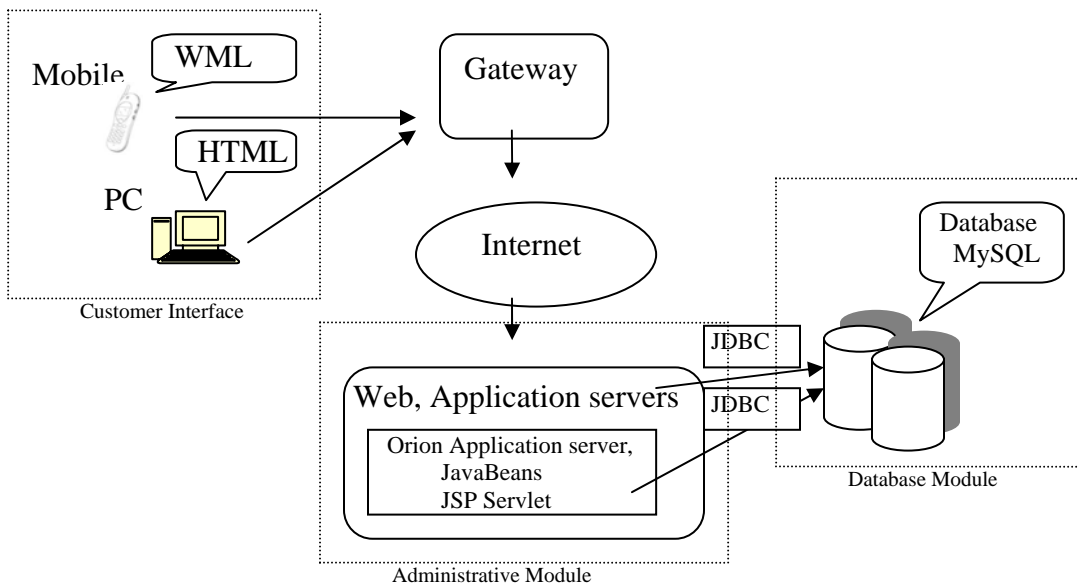


Figure 4.3 WOSS system structure

From the point of view of Software Engineering, WOSS system can also be divided into three layers such as Presentation Layer, Process Layer, and Data Layer.

**Presentation Layer** (user interface): the interfaces of mobile phone users are developed by WML and JSP. The interfaces of administrators are developed by HTML, JSP and JavaScript.

**Process Layer** (Control Layer): the orders of Mobile phone users and PC users are processed by Control Layer which designed by JSP, JavaBeans, Purchase Cart Model (Shopping Cart). Process Layer is linked to Data Layer by JDBC.

**Data Layer** (Database): product information, customer information, customer orders and any data concerned with WAP shop are saved in MySQL database. Data communication with process layer is linked by JDBC.

#### 4.3 Design of Administrative module (Web server)

According to the visualization functionalities and editing functionalities defined in Chapter 2, firstly, I appointed the functionalities of displaying, creating, editing shopping items and modifying the layout of shopping pages to ExtraWapImage.jsp which is to define the JavaScript form, table and some visual functionalities for an administrator to input the data concerned with shopping items and layout; and ewiupdate.jsp which is to retrieve the data input and execute SQL program to update MySQL database and displaying information.

Secondly, I appointed the functionalities of displaying, creating, editing product catalogs, product information and layout to PageEdit.jsp which is to define the JavaScript forms, tables and some visual functionalities for an administrator to input the data concerned with product catalogs, products and their layout; and ewipageupdate.jsp which is to retrieve the data input and execute SQL program to update MySQL database and displaying information.

Thirdly, for the developer to maintenance the system conveniently, I appointed the functionalities of deleting items, product catalogs and products to ewidel.jsp which is to retrieve the data input, delete chosen data and update the displaying information. Furthermore, to make the administrator be able to upload advertisement pictures is one of the important feathers of WOSS system, so I appointed the functionality of uploading pictures to picupload.jsp by using the component tool 'jspsmartupload.jar' which has been introduced in Section 3.4.

In the end, I appointed the functionality of the connection between web server and MySQL database to Connloader.java, as the loading file, or registration driver manager file of JDBC, which is the kernel file of connection between MySQL database and application server. It is not only used by web application server to connect to database, but also used by customer interface to load the authorization of shop users, etc.

All in all, Administrative Module (Web Server) includes seven JSP files that are able

to realize all the functionalities defined in Section 2.3.3 and Section 2.3.4. The detailed information of these JSP files can be shown as following table:

<b>File name</b>	<b>Content</b>	<b>Functionality</b>	<b>Programming language</b>
ExtraWapImage.jsp	Source code of WAP SHOP EDIT page	Display <b>WAP SHOP EDIT</b> homepage; Show the existing items (s.t wapShop); Allow the administrator to input data.	HTML, JSP, JavaScript, JDBC
ewiupdate.jsp	Source code of editing item of WAP SHOP EDIT	Create,edit items (s.t wapShop); Update displaying information and the layout of WAP shop.	HTML, JSP, JavaScript, JDBC
ewidel.jsp	Source code of delete function	Delete the existing items, product catalogs, and products; Update the displaying information.	HTML, JSP, JavaScript, JDBC
PageEdit.jsp	Source code of Page Editor page	Display <b>Page Editor</b> which shows the existing product catalogs and products; Allow the administrator to input data concerned with existing product catalogs and products.	HTML, JSP, JavaScript, JDBC
ewipageupdate.jsp	Source code editing product catalogs and products	Create, edit product catalogs and products; Update displaying information and the layout of product pages.	HTML, JSP, JavaScript, JDBC
picupload.jsp	Source code of picture upload page	Upload the advertisement pictures.	HTML, JSP, JavaScript, JDBC
Connloader.java	Source code of registration of JDBC driver manager.	Load the JDBC driver and register JDBC driver manager. Create connection between web server and database.	Java, JDBC

Table 4.1 Code files of Web server

#### 4.4 Design of Customer Interface

According to the visualization functionalities and ordering functionalities defined in Chapter 2, Firstly, I appointed the functionality of simulated customer login to wapShop.jsp which is to define the simulated customer's information, but as Section 1.3 indicated, this simulation is not realized customer validation.

Secondly, I appointed the functionalities of displaying shopping items to extraview.jsp that is to display the existing shopping items edited in WAP SHOP EDIT. Furthermore, for a developer to maintenance the system conveniently, I also appointed the functionalities of displaying product catalogs, and product lists to extraview.jsp which is also reflecting the modification from Page Editor.



Thirdly, I appointed the functionalities of saving customer single shopping order, calculating the total shopping orders, sending them to the table of MySQL database and deleting existing orders to order.jsp, saveOrder.jsp and orderDel.jsp

All in all, Customer interface module includes five JSP files that are able to realize all the functionalities define in Section 2.3.1 and Section 2.3.2. The detailed information of these JSP files can be shown as following table:

<b>File name</b>	<b>Content</b>	<b>Functionality</b>	<b>Programming language</b>
wapShop.jsp	Source code of WAP Shop homepage	Customer login page; Retrieve user information from MySQL database.	JSP, WML
extraview.jsp	Source code of displaying product catalog page, and product list page	Display the existing shopping item, product catalogs, products and submission of orders.	JSP, JDBC, WML
order.jsp	Source code of customer shopping orders	Retrieve customer's order request; Confirm order submit.	JSP, JDBC, WML
orderDel.jsp	Source code of customer deleting orders	Retrieve customer's orders; Delete request and confirm order delete.	JSP, WML
saveOrder.jsp	Source code of customer orders saving	Save and confirm customer's total orders; Send data of orders to MySQL database through JDBC.	JSP, WML, JDBC
Connloader.java (belongs to web server module, but critical usage in Customer Interface)	Source code of registration of JDBC driver manager.	Load the JDBC driver and register JDBC driver manager. Create connection between web server and database.	Java, JDBC

Table 4.2 Code files of Customer interface

In Table 4.1 and Table 4.2, I listed all the JSP files designed in WOSS system. However 'content' columns are only general description of the contents of JSP files. Readers can refer to appendix to get more detailed information of each JSP file. Readers might go through the tables associated with text description together to understand the functionalities of each JSP file.

So far, I gave the explanation of all JSP files designed in WOSS system to realize the prototype. To make the readers easily understand their correlations, it is necessary to present the following data flow chart to indicate the relations of these JSP files in different parts of WOSS system.

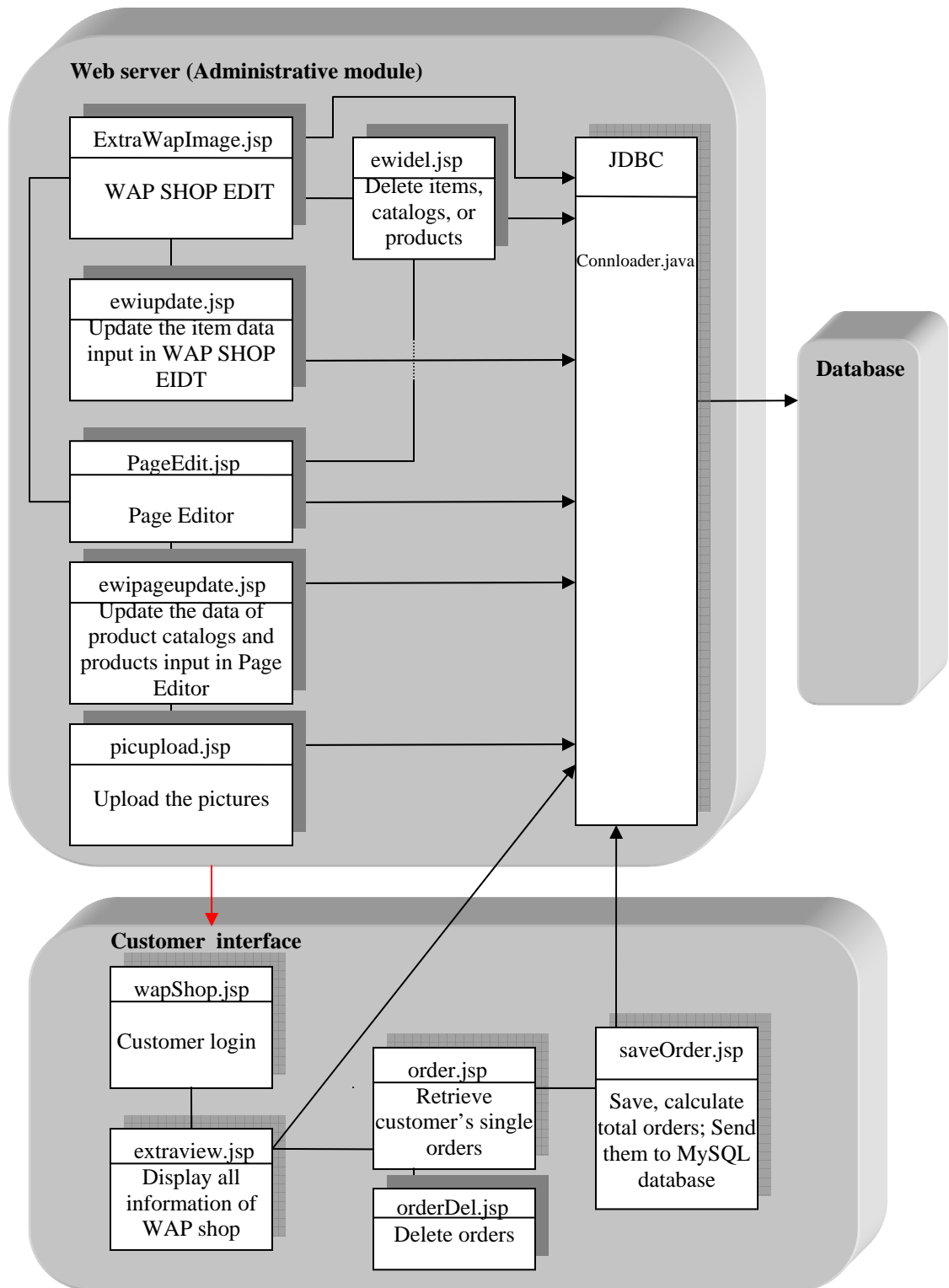


Figure 4.4 Correlations of different JSP files

From the introduction above and Figure 4.4, we find out that ExtraWapImage.jsp is responsible of displaying the existing shopping items and allowing an administrator

to edit item information and layout of WAP shop. When an administrator adds, edits, or deletes the items, it links to ewiupdate.jsp and ewidel.jsp to update the displaying information of chosen items. When an administrator wants to add, edit, or delete the product catalogs and products, it is linked to PageEdit.jsp to input new data. After he finishes information modification, it links to ewipageupdate.jsp or ewidel.jsp to update edited information. In addition, the administrator can upload the pictures by linking to picupload.jsp.

In addition, wapShop.jsp is responsible of customer identification. After a customer log into the WAP shop, it links to extrview.jsp to display existing shopping items product catalogs and product lists. When a customer wants to buy something, it links to order.jsp to get the bills and calls saveOrder.jsp to calculate his total bills and send to MySQL database. Furthermore, if a customer wants to remove his bills, it links to orderDel.jsp.

Customer Interface and Web server are all connected to MySQL database by using JDBC.

All in all, from Section 4.2 to Section 4.3, I presented the designed structure of WOSS system, applicable JSP files including their functionalities and their correlations. From these descriptions, readers may understand how I designed the WOSS system, appointed and arranged these JSP files to realize the functionalities defined in prototype. In Chapter 5 System Implementation, I will eventually present the methods and programming syntax in implementing these JSP files to realize the prototype defined in Chapter 2.

In the following section, I will introduce the design of MySQL database system by using E/R graphic analysis and important table fields of database will be discussed as well.

#### **4.5 Design of Database management system**

According to the prototype of Online Shopping System based on WAP defined in Chapter 2 and Section 2.3.5 indicated, an effective MySQL database system of WOSS system should firstly save customer data including customer personal information, and customer orders correctly; secondly, save the administrator information and system crew information including their name and ID correctly; thirdly, save the product data including item's information, the information of product catalogs and product information correctly. All these data should be saved in distinguishing tables of database individually instead of saving them in one table.

According to the mechanism of MySQL database system introduced in Section 3.3 and concept indicated above, I designed a MySQL database system of WOSS system, named it as 'gtom\_wap'.

To design an effective MySQL database system, foremost I should list the types of all data adopted in WOSS system. This can be designed as following table. Entities stands for the data groups that have several mandatory attributes in WOSS system; Attributes (fields) stand for the different data fields appointed by MySQL database management system to entities; Functionalities are to explain what these attributes stand for.

Entities	Attributes (Table fields)	Functionalities
Customer	id	Identify the customer by using his or her ID
	name	Customer's name
	gender	Customer's sex
	mobile	Customer's mobile phone number
	Details	Customer's personal information
	Reserve	Reserve for future development, for example system expansibility
Administrator	id	Identify the administrator by using his or her ID
	name	Administrator's name
	gender	Administrator's sex
	mobile	Administrator's mobile phone number
	Details	Administrator's personal information
	Reserve	Reserve for future development, for example system expansibility
System crew	id	Identify the system crew by using his or her ID
	name	The name of system crew
	gender	The sex of system crew
	mobile	The mobile phone number of system crew
	Details	The personal information of system crew
	Reserve	Reserve for future development, for example system expansibility
(Shopping) Item	id	Identify the data saved in shopping item group by using ID, for example item's ID
	text	The ID of text input for item's description
	type	The ID of position displaying, for example the new item should be displayed on 'Top' or 'bottom'.
	align	The ID of position displaying, for example the new item should be displayed on 'left' or 'right'
	url	The hyperlink ID of this item
	enter	The ID of entering a new line before displaying this item
	orderid	The ID of displaying this item's sequence, for example if wapShop's orderid is '1', and wapShop2's orderid is '2', wapShop will be displayed below wapShop2 as a descend sequence.
	pageid	The ID of this item's page
	status	The displaying status of this item, for example, if status = 1, then display this item.
	reserve	Reserve for future development, for example system expansibility
Product catalog	id	Identify the data saved in product catalog group by using ID, for example catalog's ID
	text	The ID of text input for catalog's description
	type	The ID of displaying information, for example the new catalog needs to input 'text' that displays text description on WAP shop page and Page Editor page; 'link' that displays the editing hyperlink to another Page Editor to modify this catalog; 'picture' that displays advertisement picture on WAP shop page and Page Editor page.

	align	The ID of position displaying, for example the new catalog should be displayed on 'left' or 'right'
	url	The hyperlink ID of this catalog
	enter	The ID of entering a new line before displaying this catalog
	orderid	The ID of displaying this catalog's sequence, for example if sport's orderid is '1', and DVD's orderid is '2', sport will be displayed below DVD as a descend sequence.
	pageid	The ID of this catalog's page
	status	The displaying status of this catalog, for example, if status = 1, then display this catalog.
	reserve	Reserve for future development, for example system expansibility
Product	id	Identify the data saved in product group by using ID, for example product's ID
	name	Product's name
	price	Product's price
	unit	Product's unit, for example, a DVD, or a pair of shoes
	details	The description of product information
	image	The advertisement picture of this product
	reserve	Reserve for future development
Order	id	Identify customer's order by using its ID
	userMobile	Customer's mobile phone number is identified his or her shopping order.
	productID	The ID of the product that customer wants to buy
	productName	The name of the product that customer wants to buy
	price	The price of the product that customer wants to buy

Table 4.3 Data structures of MySQL database management system in WOSS

For designing an effective database system of WOSS system, all of these data should be saved in MySQL database system correctly, but as the limitation of developing time and the large workload of this thesis, some of the data fields are only reserved for future development. For example, there is only one developer, me, who should be put into system crew and the same as system administrators, so it is not necessary to create a new table to save administrator's information and system crew's information separately. As a result I only designed one table in gtom\_wap database named as 'users' to save the data of customers, administrators and system crew. Furthermore, from Table 4.3, readers can find out, the data fields of item and product catalog are exactly same, so it's not necessary to split them into two tables. I only designed one table to save the data of items and product catalogs, named as 'extrawapimage' and the effective method to distinguish items and product catalogs is to call the different 'pageid' of the page of items or catalogs in programs and display that page in the navigator. However some of the attributes in product catalogs and items have the same name, for example 'type', but they have different meaning and they realize the different functionalities. This design can not only realize the information edit of WAP shop, but also realize the modification of layout of WAP shop. For example, 'align', 'orderid' and 'enter' can be adopted to modify the layout. The only one configuration I should do is to write different scripts in JSP file, and make them effective.

Now readers can find out how many data groups, in other words, aggregations of data employed in WOSS system and how I classify them. To make the readers easily understand the attributes of entities list in Table 4.3 and the correlations between these data aggregations, I introduce the E/R analysis algorithm which can be visualized in Figure 4.5. In this figure, the rectangle stands for the data aggregations or entities in WOSS system; the oval stands for the attributes, in other words, the table fields of those data aggregations; and the lozenge stands for the decision or active correlation between different entities.

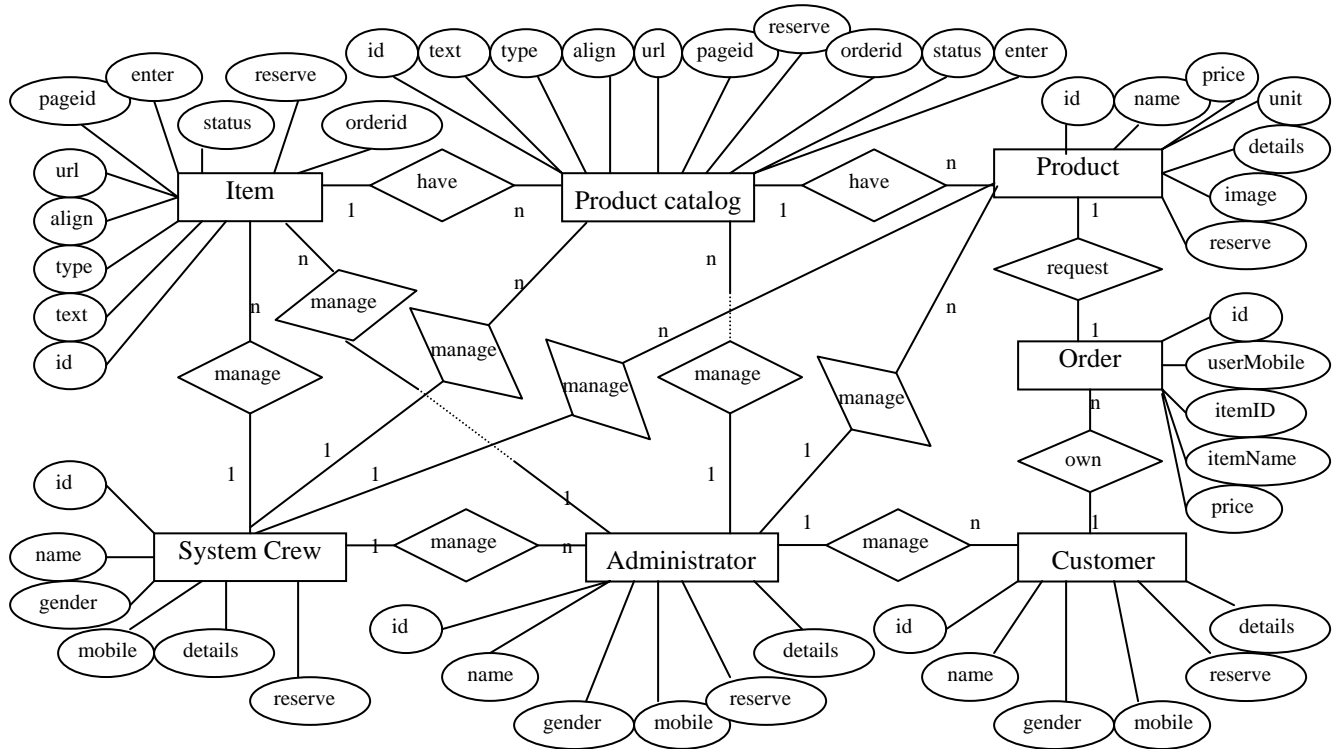


Figure 4.5 E/R graphical analysis of data structures in WOSS system

Figure 4.5 lists the correlation between different entities and the attributes described in Table 4.3. Firstly, I assume there is only one system crew in WOSS system, and one system crew can manage the data of n administrators. In addition, one administrator can manage the data of n customers. Secondly, one customer can own n orders. But one order only requests by a product. Thirdly, one item can have n product catalogs. Similarly a product catalog can have n products. In the end, both a system crew and an administrator are able to manage the data of n items, a product catalogs, and n products. This figure is adopted to guide the design of MySQL database system of WOSS.

As the concept indicated above, I designed gtom\_wap which has 5 different tables to save different kinds of data corresponding Table 4.3. They are:

**extrawapimage:** saving shopping item data, product catalog data and all the data concerned with WOSS system demo

**products:** saving the data of products sold in WAP shop

**users:** saving the data of the customer personal data, the administrator personal data and system crew personal data

**userorder:** saving the data of customer shopping orders

**showwindows:** saving extensive usage data of WOSS system for further development, for example, the better displaying data of WOSS system.

The terms with underlines stand for the entities list in Table 4.3 and Figure 4.5. They are classified into above designed tables. These five tables of gtom\_wap are saving all the data described in Table 4.3, and to find out the correlations between these five tables, readers can refer to Figure 4.5.

So far I have introduced the design information of MySQL database in WOSS system, including the data aggregations, data fields defined in tables and their correlations. For detailed implementation information, in other word, if the readers want to find out how I can create these tables in MySQL database management system, readers can refer to Section 5.3 Implementation of MySQL database.

## 4.6 Conclusion

In this Chapter, I explained the design details of WOSS system, for example web server design, customer interface design and database design. To make readers easily understand the structure, and mechanism of WOSS system, I gave the explanation of correlations of JSP files data structure analysis of database system by using data structure table and E/R analysis. After reading this chapter, readers might get an all-sides comprehension how I organized WOSS system. In next chapter, I will present the detailed implementation information including all three major modules of WOSS system.

## 5 System Implementation

In this chapter, the implementation of WOSS system is presented. In order to make readers easily understand the implementation of WOSS system, I give the description followed by the prototype defined in Chapter 2 and the system structure presented in Chapter 4. The implementation details are presented in Customer Interface, Administrative Module (Web server) and MySQL database system. Since the source codes are listed in appendix and the installation of development tools has already been stated at Chapter 3, I only quoted the most important part of JSP files to proceed the description of system implementation and the installation processes are not repeatedly discussed in this chapter.

According to the prototype defined in Chapter 2, the WOSS system should be implemented with the visualization functionalities of both customer interface and administrative module; the ordering functionalities of customer interface; editing functionalities of administrative module and the functionalities of MySQL database system. In section 5.1 I will present the implementation of administrative module, in section 5.2 I will present the implementation of customer interface and in section 5.3 I will present the implementation of MySQL database system. All these sections are approached from the functionalities of the prototype. To give readers a visual and live description, I will also adopt some system screenshots.

### 5.1 Implementation of Administrative module (Web server)

As Chapter 4 System Design indicated, administrative module (web server) is the most important part of WOSS system. It governs the whole WOSS system, for example, adding new products, edit products' information and upload the pictures. So I introduce the implementation of this module instead of giving the introduction of implementation of customer interface first.

Foremost, it's very important to implement this module easily to be used, and understandable to the future system users. The combination of JSP, HTML, JavaScript and SQL is one of the significant features in this module. I give the explanations of typical JSP files of this module, instead of repeating the explanation of all, because of syntax similarities.

As Chapter 2 indicated that administrative module should have the functionalities of editing shopping items, product catalogs, products and the layouts of WAP shop. According to these functionalities, in Chapter 4, I designed this module as two major interfaces, WAP SHOP EIDT which is responsible for editing shopping items; and Page Editor which is responsible for editing product catalogs, products, and the layout of WAP shop. Since the functionalities of these two interfaces are almost same, nothing but to create, add, edit and delete data of shopping items, product catalogs and products, and modify the layouts of item's webpage, catalog's webpage and



product's webpage, I implemented the entire functionalities of data creating, editing, deleting and layout modifying to both WAP SHOP EDIT and Page Editor. This is the reason why the appearances of these two interfaces that will be shown in Figure 5.1 and Figure 5.2 are almost same. The detailed explanation of functional implementation of Page Editor will come up in Section 5.1.2.

### 5.1.1 Implementation of ExtraWapImage.jsp (WAP SHOP EDIT)

'ExtraWapImage.jsp' is the homepage of administrative module. After initiating the services of Orion application server (to find out how to start it, please consult Section 3.4), administrators can input <http://localhost/ExtraWapImage/ExtraWapImage.jsp> to enter this page which is named as 'WAP SHOP EDIT'. Administrator should find this webpage on screen:

ID	TYPE	ITEM TITLE	POSITION	EDIT	ENTER NEW LINE	ARRANGE SEQUENCE	PAGE NUMBER	STATUS	EDIT DELETE
2	top	wapShop	NUL	<a href="#">EDIT</a>	NUL	1	0	<input checked="" type="checkbox"/>	
42	top	Test	NUL	<a href="#">EDIT</a>	NUL	1	0	<input type="checkbox"/>	
<a href="#">CREATE NEW ITEM</a>	HOME PAGE		NUL	NUL	NUL	1	Automatically Distribute	<input type="checkbox"/>	<a href="#">Add this ITEM</a>

Figure 5.1 Screenshot of WAP SHOP EDIT homepage

In WAP SHOP EDIT, 'ID' stands for the shopping item's ID which is distributed by MySQL database automatically. When an administrator input a new item, the database system will assign a new ID to this item. This ID seems randomly. It is because the 'id' defined in extrawapimage table of MySQL database does not only stand for the item's ID but also represent all the data's ID saved in extrawapimage table. And it's defined as 'auto\_increment' which means MySQL database will point the ID to data in sequence. 'ID' can be modified in extraview.jsp to display this item

'TYPE' stands for the displaying position of created items on navigator. As Figure 5.1 shows, the create item wapShop that is for system demo is on top. 'POSITION' stands for displaying position of items as well, but it is to set the item to left or right. 'ENTER NEW LINE' stands for inserting a new enter (a blank row) between two items, since the item should be displayed on WAP shop page individually, the default value of this field is 'NUL'. These three can accomplish the functionalities of editing shopping item pages of WAP shop. 'EDIT' stands for the hyperlink to Page Editor for editing the product catalogs of this item. For a number of shopping items, WOSS system can allow the administrator to arrange priority sequence to them. This is the functionality of 'ARRANGE SEQUENCE' and this sequence is implemented for displaying the items in WAP SHOP EDIT, for example, if I set the sequence of 'test'

as 2, 'test' should be displayed above 'wapShop' as the higher priority than 'wapShop'. This accomplishes the functionalities of editing the layout of WAP SHOP EDIT. 'PAGE NUMBER' stands for the page ID of product catalogs of this shopping item. Different product catalogs of one shopping item have the same 'PAGE NUMBER' and this 'PAGE NUMBER' is not shown in WAP SHOP EDIT but only displayed when an administrator go into the Page Editor to edit product catalogs. The detailed information of 'PAGE NUMBER' will be given in next section. 'STATUS' stands for the displaying status of this item, as Section 2.3.4 indicated, if an administrator tick this status off, it means this item will not be displayed in WAP shop page. 'EDIT DELETE' stands for the option to an administrator. For example if an administrator wants to change the name of this item, first he inputs new name replaced the previous one under 'ITEM TITLE' and clicks on 'EDIT' to update this item's name, and if he wants to delete this item, he just simply clicks on 'DELETE' to remove this item.

If an administrator wants to add a new shopping item, first he should input the item's name in the blank located in the last row, right behind 'CREATE NEW ITEM' shown in Figure 5.1, and clicks on 'Add this ITEM'.

So far, I have given the description of elements of WAP SHOP EDIT. The implementation of these elements can realize the prototype functionalities defined in Chapter 2 for visualization of homepage of administrative module and functionalities of editing shopping items. Afterwards, I give the description of my programming implementation in creating the table in WAP SHOP EDIT, defining the JavaScript forms, and hyperlink button to Page Editor. This introduction is following the structure of Figure 5.1 step by step.

**First, define the entire table in WAP SHOP EDIT:**

```

<form name="sort"
... .. // beginning of form definition, define the form name as 'sort'

<table border="8" bgcolor="#F0F8FF" cellspacing="0" cellpadding="0" width="100%">
<tr>
<td height="42" colspan="10" valign="middle">
<div align="left"><font color="#5f8ac5">
<font>WAP SHOP EDIT</font>
</div>
</td>
</tr>
// <div>...</div> is text alignment which stands for a block-level element, simply
// defines a block of content in the page

```

**Second, define the table's elements of first row including Text output:**

```

tr>
<td height="27" align="center"><font color="#999999">ID</font></td>

```

```

<td height="27" align="center"><font color="#999999">TYPE</font></td>
<td height="27" align="center"><font color="#999999">ITEM TITLE</font></td>
<td height="27" align="center"><font color="#999999">POSITION</font></td>
<td height="27" align="center"><font color="#999999">EDIT</font></td>
<td height="27" align="center"><font color="#999999">ENTER NEW LINE</font></td>
<td height="27" align="center"><font color="#999999">ARRANGE SEQUENCE</font></td>
<td height="27" align="center"><font color="#999999">PAGE NUMBER</font></td>
<td height="27" align="center"><font color="#999999">STATUS</font></td>
<td height="27" align="center"><font color="#999999"><nobr>EDIT
DELETE</nobr></font></td>
// All text between the start and end of the NOBR elements will not have line
// breaks inserted between them.
</tr>

```

**Third, define an Input Form and a Display Form of second row:**

```

<tr>
<td align="center"><%=rs.getString("id")%></td>

<td align="center"><%=rs.getString("type")%><input type=hidden
name=type<%=rs.getString("id")%>
value="<%=rs.getString("type")%>"></td>

<td align="center"><input type=text name=text<%=rs.getString("id")%>
value="<%=rs.getString("text")%>"></td>

<td align="center">NUL<input type=hidden
name=align<%=rs.getString("id")%> value="NUL"></td>
.....

```

**Fourth, define a hyperlink Button linking to PageEdit.jsp, which is displayed as 'EDIT' in fifth column in the Figure 5.1:**

```

<td align="center">
<a href="PageEdit.jsp?PAGEID=<%=rs.getString("url")%>">EDIT</a><input
type=hidden
name=url<%=rs.getString("id")%>value="<%=rs.getString("url")%>"></td>
// send the page ID to Page Editor

```

Since some important programming implementations are also adopted in Page Editor, for example, to define a checkbox in JavaScript form, create an alert box and so on, I will give these descriptions in next section.

### 5.1.2 Implementation of PageEdit.jsp (Page Editor)

As I mentioned in the beginning of Section 5.1, Page Editor is responsible for editing the information of product catalogs and products. Readers may have the question that why I combined these two functionalities, editing product catalogs and editing products, together, and in the prototype defined in Chapter 2, administrative module should have the separated functionalities of editing product catalogs and products.

Actually, the visualization of web page of WAP shop product catalogs is quite similar as the page of product list. They all list some titles no matter what the titles are either product catalogs or products. The displaying methods of them are exactly same. The only one difference is in product page, it should display the price information and order options. As a result, I implemented a general Page Editor which has the functionalities of both editing product catalogs and products. An administrator can simply proceed the process of editing product catalogs without choose some functionalities reserved for editing products, for example the option of 'order'. An administrator doesn't need to choose this option when he edits product catalogs.

After an administrator adds a new item, he click on 'EDIT' introduced in the last section, the navigator should link to the following web page, named Page Editor:











-----Page Editor <a href="#">Back to Homepage</a>									
ID	TYPE	DESCRIPTION	POSITION	LINK	ENTER NEW LINE	ARRANGE SEQUENCE	PAGE NUMBER	STATUS	EDIT DELETE
3	pic		center		<input checked="" type="checkbox"/>	100	2	<input checked="" type="checkbox"/>	 
10	link	Sports	center	<a href="#">Sports</a>	<input checked="" type="checkbox"/>	3	2	<input checked="" type="checkbox"/>	 
8	link	new DVD	center	<a href="#">new DVD</a>	<input checked="" type="checkbox"/>	2	2	<input checked="" type="checkbox"/>	 
6	link	new Phone	center	<a href="#">new Phone</a>	<input checked="" type="checkbox"/>	1	2	<input checked="" type="checkbox"/>	 
36	link	new books	center	<a href="#">new books</a>	<input checked="" type="checkbox"/>	1	2	<input checked="" type="checkbox"/>	 
CREATE NEW CONTENT	PICTURE		CENTER	NUL	<input type="checkbox"/>	1	2	<input type="checkbox"/>	<a href="#">ADD NEW CONTENT</a>

Figure 5.2 Screenshot of Page Editor for product catalogs

As the Figure 5.2 displayed, 'ID' represents for all data ID saved in extrawapimage table of MySQL database. 'TYPE' stands for the classification of input data in second column, the last row of table. It has four options which can be chosen in the checkbox of second column, the last row. The options are: 'PICTURE' which is for uploading the advertisement pictures of product catalogs; 'TEXT' which is for labeling the text description of product catalogs; 'LINK' which is for inputting product lists and creating hyperlink to another Page Editor to edit this product. 'ORDER' as I mentioned in the beginning of this section, it's reserved for editing product's price.

'LINK' stands for the link to another Page Editor to edit the product information of this product catalog.

'ENTER NEW LINE' stands for insert a new blank row between two product catalogs. 'ARRANGE SEQUENCE' stands for displaying priority sequence of these catalogs. If an administrator sets one catalog's sequence as '2', this catalog will be displayed above another catalog whose sequence is '1'. 'POSITION' stands for displaying position of this catalog on WAP Shop page. It has three options 'center'; 'left'; 'right'. These three can accomplish the visualization functionalities defined for product catalog pages.

'PAGE NUMBER' stands for the displaying the page ID of this catalog. All product catalogs, including their contents, for instance the advertisement pictures and text descriptions, etc of one shopping item have the same page ID or 'PAGE NUMBER'. 'PAGE NUMBER' (page ID) is inherited from 'ID'. For example, when an administrator adds a new product catalog, database will appoint an 'ID' to this product catalog, so the page ID of this catalog is identical to its 'ID' and all the data including pictures and enclosed products of this catalog has the same page ID. Furthermore, when an administrator adds a product to this catalog, database appoints an 'ID' to this product which has the same page ID to its catalog's, and the page ID of this product's contents is identical to its 'ID'. This mechanism guarantees the classified data of one item are displayed in the identical webpage, and the classified catalogs or products are displayed in the identical webpage as well. Readers can consult Figure 5.1, Figure 5.2, and Figure 5.3 to get a comprehensive understanding. For example the 'ID' of 'wapShop' is '2', so the pictures and all the catalogs of this item has the same page ID '2'. Furthermore, the 'ID' of catalog 'sports' is '10', so its pictures and all the products will have the same page ID '10'

'STATUS' stands for the visual option of product catalogs. It's same as in WAP SHOP EDIT. If one doesn't want display any catalog, just tick off the checkbox in this catalog's row. 'EDIT DELETE' has the same functionalities as in WAP SHOP EDIT.

If an administrator wants to add a new product catalog, he should firstly input the title of this catalog in the blank of 'DESCRIPTION' and choose 'LINK' as its type. Secondly, he should press the hyperlink on 'ADD THIS CONTENT' and tick the status on. As above introduction, he can use 'ENTER NEW LINE' and 'ARRANGE SEQUENCE' to change the layout of this catalog page. If he wants to add some text description or advertisement pictures, he simply input the words in the blank of 'DESCRIPTION' and chooses 'TEXT' as its type. Afterwards he can upload the picture by choosing 'PICTURE' type.

So far I gave the complete description of Page Editor for editing product catalogs. In the following I introduce the Page Editor for products. The screenshot of Page Editor for product is shown as following figure:

Page Editor <a href="#">Back to Homepage</a>									
ID	TYPE	DESCRIPTION	POSITION	LINK	ENTER NEW LINE	ARRANGE SEQUENCE	PAGE NUMBER	STATUS	EDIT DELETE
11	link	tent	center	<a href="#">tent</a>	<input checked="" type="checkbox"/>	10	10	<input checked="" type="checkbox"/>	
13	link	water bottle	center	<a href="#">water bottle</a>	<input checked="" type="checkbox"/>	8	10	<input checked="" type="checkbox"/>	
CREATE NEW CONTENT	PICTURE		CENTER	NUL	<input type="checkbox"/>	1	10	<input type="checkbox"/>	<a href="#">ADD THIS CONTENT</a>

Figure 5.3 Screenshot of Page Editor for products

From Figure 5.3, we can find out the layouts, and functionalities of this page are exactly the same as Page Editor for product catalogs and all products of one product catalog have the same 'PAGE NUMBER' (page ID) which has the same functionality as I explained above. So I will not repeat the description for this page again.

All in all the implementation of Page Editor can accomplish the prototype functionalities defined in Chapter 2 both in visualization and editing. Afterwards, I give the description of my programming implementation of Page Editor. This introduction is following the structure of Figure 5.2, Figure 5.3 step by step.

**First, define a Dropdown list which is displayed in seventh column in Figure 5.3 when an administrator needs to arrange the descending sequence to highlight the displaying priority:**

```

<td align="center">
<select name="orderid"<%=rs.getInt("id")%>">
<option value="<%=rs.getString("orderid")%>"selected"><%=rs.getString("orderid")%>
</option>
<%
for(int i = 1; i < 10 ;i++)
// create a 1 to 10 circle, instead of writing all the number from 1 to 10
... ..
</select></td>

```

**Second, define a Checkbox which is displayed in ninth column in Figure 5.3 to confirm the status of displaying a product catalog or a product. In other words whether they will be displayed in customer shopping page or not:**

```

<td align="center"><%=rs.getString("pageid")%></td>
<td align="center">
<input type="checkbox"
name="status"<%=rs.getString("id")%>"onclick="changechecked(this)"
// call function changechecked (), which is explained later
<%
if (rs.getInt("status") == 1){
out.print("value='on'");
out.print("checked");

```

```

}
// display the status of checkbox is ticked
%>
</td>

```

**Third, define an Alert box which is displayed in the last column in Figure 5.3 when an administrator's cursor of mouse clicks on 'Edit' or 'Delete':**

```

<td align="center">
<a href = "#" onclick="quickmodify (<%=rs.getInt("id")%>)"></a>&nbsp;
// Picture saved at orion\default-web-app\WapAdmin\images\new
<a href = "ewidel.jsp?ID=<%=rs.getString("id")%>&FROM=EWI"></a>
</td>
</tr>
// '&nbsp;'23 is the entity used to represent a non-breaking space.

```

**Fourth, define a Checkbox update button which is displayed as 'ADD THIS CONTENT' in the last column in Figure 5.3, When an administrator creates a product catalog or a product:**

```

td align="center">
<a href = "#" onclick="quickmodify (0)">ADD THIS CONTENT</a>
</td>
</tr>
// call the function 'quickmodify ()', which will be explained later.
</table>
</form>
// finishing form definition

```

**Fifth, define the function changechecked(checkbut), to judge the status of checkbox**

```

function changechecked(checkbut){
var v =checkbut.checked;
if (v) { checkbut.checked = true;    }
}

```

**Sixth, define the function quickmodify (id), to check the data of whole form and send the data to other JSP files and update the database.**

---

<sup>23</sup> &nbsp;: [www.sightspecific.com/~mosh/WWW\\_FAQ/nbsp.html](http://www.sightspecific.com/~mosh/WWW_FAQ/nbsp.html), LoneWolf, 1993-2003

```

function quickmodify(id){
var f = document.forms["sort"];
// corresponding defined form 'sort'
var v =id;
document.forms ["sort"].elements ["id"].value=v;
document.forms["sort"].elements["type"].value=document.forms["sort"].element
s["type"+id].value;
document.forms["sort"].elements["text"].value=document.forms["sort"].elements
["text"+id].value;
document.forms["sort"].elements["align"].value=document.forms["sort"].elemen
ts["align"+id].value;
document.forms["sort"].elements["url"].value=document.forms["sort"].elements[
"url"+id].value;

if (document.forms["sort"].elements["enter"+id].checked)
    document.forms ["sort"].elements ["enter"].value=1;
    else
        document.forms ["sort"].elements ["enter"].value=0;

document.forms["sort"].elements["orderid"].value=document.forms["sort"].elem
ents["orderid"+id].value;

if (document.forms["sort"].elements["status"+id].checked)
    document.forms ["sort"].elements ["status"].value=1;
    else
        document.forms ["sort"].elements ["status"].value=0;

// check and retrieve the data of form

var surl =
"ewiupdate.jsp?ID="+v+"&TYPE="+document.forms["sort"].elements["type"].
value+"&TEXT="+document.forms["sort"].elements["text"].value+"&ALIGN="
+document.forms["sort"].elements["align"].value+"&URL="+document.forms["
sort"].elements["url"].value+"&ENTER="+document.forms["sort"].elements["e
nter"].value+"&ORDERID="+document.forms["sort"].elements["orderid"].valu
e+"&STATUS="+document.forms["sort"].elements["status"].value;
    window.open (surl, "_self");
}
// activate ewiupdate.jsp to send data to other JSP files and update database

```

**Seventh, define the Picture Upload functionality to allow the administrator to upload advertisement picture.**

The screenshot of picture upload can be shown as following picture:



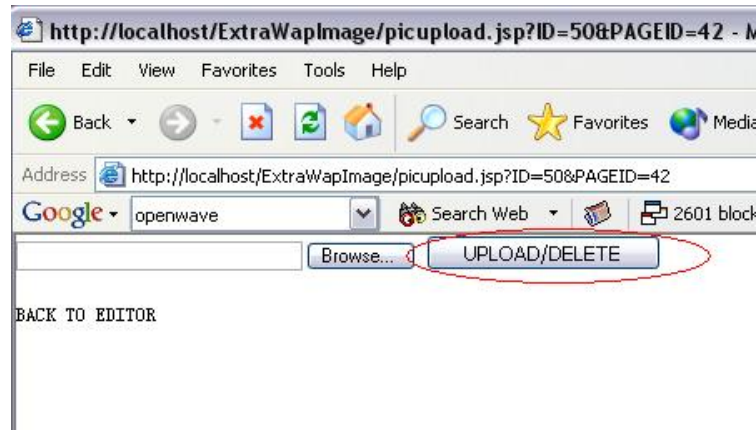


Figure 5.4 Screenshot of Picture Uploading

I implemented the following script to realize this functionality.

```

if(rs.getString("type").equals("pic")){
    String ss="UPLOAD PICTURE";
    .....
    href="picupload.jsp... // link to picupload.jsp

```

In Chapter 4 system design I introduced picupload.jsp is responsible for uploading pictures, the kernel implementation of this JSP file is the component tool, jspsmartupload.jar of Orion application server is adopted.

Now the readers might understand how I organized and implement WAP SHOP EDIT and Page Editor. In the following, I give the description of partial SQL query implementation in Administrative Module to let the readers understand how WAP SHOP EDIT edits and retrieves the data.

### 5.1.3 Implementation of data query and retrieving the data to display by using JavaScript and SQL

In WOSS system, all the data search, data modify, data insert or data update are executed by SQL. They construct the functionalities of creating new products catalogs, products information inputting, item, products information editing, etc.

As in Chapter 4 system design, I introduced that ewiupdate.jsp is responsible for retrieving data input in WAP SHOP EDIT and update the data. The general implementation of retrieving data is:

Define the methods of 'ewiupdate.jsp' retrieving the parameters influx from above:

```

String id=request.getParameter ("ID");
String type=request.getParameter ("TYPE");
String text=request.getParameter ("TEXT");

```

```
String align=request.getParameter ("ALIGN");
String url=request.getParameter ("URL");
String enter=request.getParameter ("ENTER");
String orderid=request.getParameter ("ORDERID");
String pageid=request.getParameter ("PAGEID");
String status=request.getParameter ("STATUS");
```

JSP should communicate each other by sending parameters. After retrieving the parameters, one JSP file can correctly use the data from another JSP file to finish its function. JSP file retrieves parameter influx, e.g page 'ID', and display this page, for instance in ExtraWapImage.jsp:

```
String id=request.getParameter ("ID");
// Retrieve page 'ID' from database
String htmltitle="WAP SHOP EDIT";
String PAGES=request.getParameter ("PAGES");
```

Define the SQL insert program and update/edit program by using stat.executeUpdate(sql) method:

```
if(!id.equals("0")){
    sql="UPDATE ExtraWapImage SET
    type='"+type+"',text='"+text+"',align='"+align+"',url='"+url+"',enter="+ent
    er+",orderid="+orderid+",status='"+status+" WHERE ID="+id;

    stat.executeUpdate(sql); // edit the data by using stat.executeUpdate(sql)
    else {
        sql="insert into ExtraWapImage(type,text,align,enter,orderid,status)
        values('top','"+text+"','NUL','NUL','"+orderid+"','"+status+")";
        // insert data by using stat.executeUpdate(sql)
        stat.executeUpdate(sql);
    }
```

According to retrieved parameters, define the SQL program, and execute the SQL query by stat.executeQuery( sql ), for instance in ExtraWapImage.jsp:

```
if (pages==0)
    sql="select * from ExtraWapImage where type='top' order by orderid desc";
    else sql="select * from ExtraWapImage where pageid="+pages+" order by
    orderid desc";
// execute SQL searching program by using stat.executeQuery(sql)
rs=stat.executeQuery( sql );
```

The complete SQL query and insert program is, for instance in extraview.jsp:

```
sql="select * from ExtraWapImage where type='top' and text='"+text+" and
orderid="+orderid;
```

```

    ResultSet rs=stat.executeQuery(sql);
    rs.next();
    // search the data from ExtraWapImage table

    sql="UPDATE ExtraWapImage SET url='"+rs.getString("id")+"',pageid=0
    where id='"+rs.getString("id");
    stat.executeUpdate(sql);
    }
    // and insert the data into 'url'

```

To get the query results, I use 'rs.getString(), rs.getInt() or getDate()' method. Detailed information of getXXX() method, please consult Section 3.6.

Since SQL syntax is adopted in implementation of Administrative Module comprehensively and they are located in JSP files dispersedly, I cannot present the detailed description of every SQL query sentence in this section. I only quoted the most typical and important parts of SQL. To get more information of implementation of SQL query, readers can refer to appendix. In the next section, I will introduce JDBC.

#### 5.1.4 Implementation of JDBC

In Chapter 4 system design, I give the explanation of WOSS system structure. One of the most important parts of Administrative Module is the connection between web serve and database system. So how to implement the JDBC connector correctly is very important to realize the prototype defined in Chapter 2. In Section 3.6 I introduced the basic steps of construction of JDBC, in this section I present the steps of implementing JDBC in WOSS system in details.

As a prerequisite to each, the first step is to load or register the database driver (in WOSS system I have loaded the driver of MySQL) with DriverManager, by creating the file named ConnLoader.java which is legally located at orion\default-web-app\WEB-INF\classes\com\ in Orion application server, this directory is specifically designed by Orion application for saving JDBC driver and will be called by ExtraWapImage.jsp; extraview.jsp; wapShop.jsp; order.jsp; saveOrder.jsp, etc.

The DriverManager needs to be told which JDBC drivers it should try to make connections with. And In ConnLoader.java, I used Class.forName() on the Class that implements the java.sql.Driver interface. For instance in ConnLoader.java (Appendix 8.2.12):

```

package com.gtom.wap;
import java.sql.*;
.....
Class.forName("org.gjt.mm.mysql.Driver").newInstance();

```

```
// newInstance() is Creates a new instance of the class.
```

And the documentation of MySQL JDBC driver states the Class name to use, which is 'org.gjt.mm.mysql.Driver'

Once the driver is registered, a connection should be established. Obtaining a Connection requires a URL for the database. This URL is constructed using the following syntax, where items contained in square brackets are optional:

```
jdbc:mysql://[hostname][:port]/dbname[?param1=value1][&param2=value2]...
```

After statement of URL of database, it is passed to the DriverManager.getConnection() method to obtain a Connection Object. In ConnLoader.java of WOSS system, I wrote it as:

```
conn = DriverManager.getConnection(  
"jdbc:mysql://127.0.0.1/gtom_wap?useUnicode=true&characterEncoding=gb2  
312&user=root&password=");
```

After above process, the DriverManager class, in other words, the connection to database has been established. The second step is to call this DriverManager in JSP file and establish the connection between JSP and MySQL database through JDBC. For instance, in ExtraWapImage.jsp (Appendix 8.2.1), I wrote the following declaration tag in the beginning:

```
<%@ page import="java.sql.*" %>  
<%@ page import="com.gtom.wap.*" %>
```

```
// to call the DriverManager located at orion\default-web-app\WEB-  
INF\classes\com\gtom\wap, and packed as com.gtom.wap.
```

Since the DriverManager, ConnLoader.java will be called by several JSP applications; the code below should be inserted into JSP scriptlets every time.

```
Connection Conn = ConnLoader.getInstance().getConnection(this);  
// to establish the connection to database by calling ConnLoader (the driver manager)
```

After the connection establishment, a JDBC statement object which sends the SQL query to MySQL database management system. It takes an instance of active connection to create a statement object, by writing 'conn' as the connection object and 'stat' as the statement creation. For instance in ExtraWapImage.jsp (Appendix 8.2.1)

```
Statement stat = Conn.createStatement();
```

This driver file is called every time when Web server applications need communicate with MySQL database, by using:

```
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*" %>
<%@ page import="com.gtom.wap.*" %>
// load JDBC driver, defined in JSP scriptlet tag
.....
<%Connection Conn = ConnLoader.getInstance().getConnection(this);
    Statement stat = Conn.createStatement(); %>
// acquire JDBC connection
```

Close the MySQL database link in a JSP file which finishes data operation. Although there is not only one database link, but after data operation in a single JSP file, one should close it to save database resource. By using:

```
<% if(rs!=null)rs.close();
    //conn.freeConnection();
    Conn.close();
%>
```

To read the complete source code, readers can refer to Appendix 8.2.12.

## 5.2 Implementation of Customer interface

As the prototype defined in Chapter 2 indicated, Customer Interface is to display the existing shopping item, product catalogs and product information; to save the customer's orders, and to record them in MySQL database through web server. In this section, I will approach my description of implementation of Customer Interface from visualization functionalities and ordering functionalities as those defined in Section 2.3.1 and Section 2.3.2.

I will discuss wapShop.jsp and extraview.jsp to present the implementation of visualization functionalities, and I will discuss saveOrder.jsp to present the implementation of ordering functionalities.

Since the WAP shop pages should be displayed on the navigator that supports WAP, I use Opera navigator as my default shopping browser of WAP shop. Parts of source code will be quoted in this section, but the complete JSP files will be listed in appendix.

### 5.2.1 Implementation of wapShop.jsp and extraview.jsp

'wapShop.jsp' is to simulate the WAP shop customers login page, as Section 1.3 Correlated Work indicated, the identity authorization functionality has not been realized in WOSS system yet, so this page is only the simulation trail. I defined the session object to save customer logging on information as following code shown (complete source code referred to Appendix 8.2.7):

```
<card id='welcome' title='welcome'>
  <p>
  Welcome
  </p>
  <%
    session.setAttribute("name","Mike");
    session.setAttribute("phone","13641366774");
    session.setAttribute("address","2nd east street.");
    response.sendRedirect("extraview.jsp");
  %>
  <do type="options" label="home"><go
  href="http://localhost/ExtraWapImage/extraview.jsp" /></do>
  <do type="prev" label="back"><prev /></do>
</card>
```

After a customer finish his or her identity validation, the navigator will jump to extraview.jsp that displays the shopping item WAP SHOP EDIT edited, for example the shopping item, wapShop created in WAP SHOP EDIT, and it consists of product catalogs. The screenshot of factual customer shopping is shown as following figure:

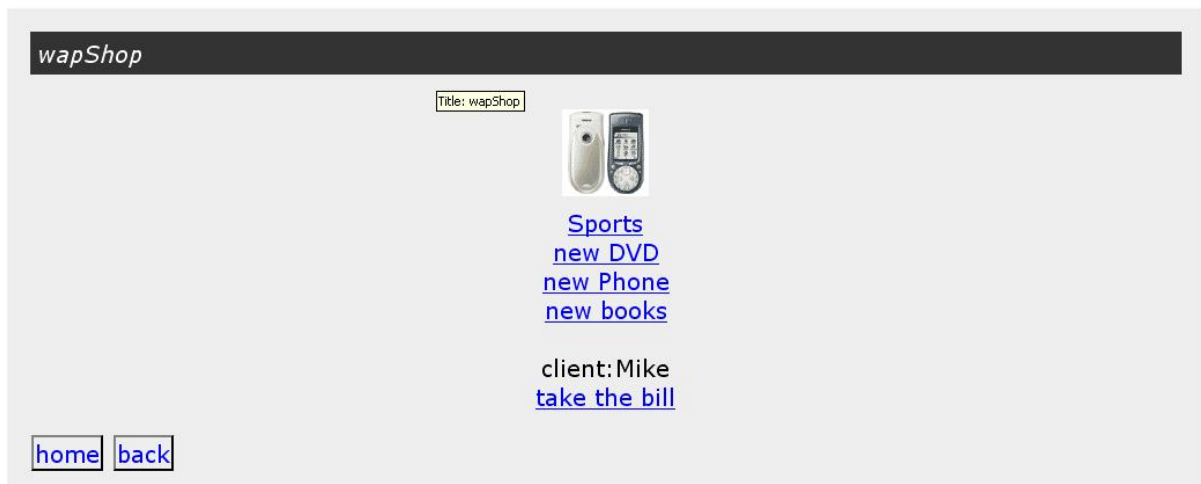


Figure 5.5 Screenshot of Customer shopping page

After a customer chooses a product catalog, for instance 'Sports', he simply clicks on the hyperlink of 'Sports'. Afterwards, the navigator links to 'Sports' the product catalog page. The screenshot is shown in following figure:



Figure 5.6 Screenshot of the product catalog 'Sports'

From Figure 5.6, we can find out that there are two products in 'Sports' catalog. They are tent and water bottle which added in Page Editor. Readers can refer Figure 5.3 to understand the correlations between Page Editor and this catalog page. If the customer chooses tent as his expected product to buy, he simply clicks on the hyperlink of 'tent' and the navigator links to product page. The screenshot is shown as following figure:



Figure 5.7 Screenshot of the product 'Tent'

This implementation accomplishes the visualization functionalities defined in Chapter 2 and the hierarchy of these pages is perfectly identical to Figure 2.2. Now I introduce the programming implementation of Customer Interface.

**First**, 'extraview.jsp' retrieves the parameter of the item's ID, creates the JDBC connection to MySQL database, and verifies this item's ID to display this item in navigator as Figure 5.5:

```
String id = request.getParameter("ID");
```

```

if(id==null) id = "2";
Connection Conn = ConnLoader.getInstance().getConnection(this);
Statement stat = Conn.createStatement();
String sql = "select * from ExtraWapImage where id="+id;
ResultSet rs = stat.executeQuery(sql);
.....

```

**Second**, the following code of extraview.jsp is to display the layout of shopping contents, and judge if it needs to make a new paragraph in shopping webpage.

```

boolean p = false;
boolean pp = false;
sql = "select * from ExtraWapImage where pageid="+id+" and status=1 order
by orderid desc";
// retrieve the pageid (PAGE NUMBER) and orderid (SEQUENCE) from
//database and display the contents descendingly.

rs = stat.executeQuery(sql);
String align=null;
while(rs.next()){
    if(align==null||!align.equals(rs.getString("align"))p=true;
// adopt variable p to judge if it needs to modify the 'align' of contents
// adopt variable pp to judge if it needs to make a new paragraph
    if(p){
        if(pp)out.println("</p>"); // if pp == true, make a new paragraph
        out.println("<p align='"+rs.getString("align")+"'>");
        p=false;
        pp=true;    }
    align=rs.getString("align");
.....

```

As in Section 5.1.2, I have explained that ‘TYPE’ includes ‘PICTURE’ ‘LINK’ ‘TEXT’ and ‘ORDER’. The methods adopted to display these contents in the customer interface can be concluded as following:

**Display graphic content, such as the picture of two mobile phones shown in Figure 5.5:**

```

if(rs.getString("type").equals("pic")){
// use graphic method to display this picture, url stands for picture location
out.println("<img src='"+rs.getString("url")+''
alt='"+rs.getString("text").trim()+''/>";    }
// When customers browse the picture contents, the WAP cell phone needs time to
//download the picture first. To optimize this delayed time, WAP cell phone will
//display this 'text' description first.

```



**Display text content, such as the text description ‘beautiful and warm’ shown in Figure 5.7:**

```
if(rs.getString("type").equals("text")){  
  //use text method to display text content  
  out.println(rs.getString("text").trim()); }  
}
```

**Display hyper link content, such as the hyperlink of ‘sports’ shown in Figure 5.5:**

```
if(rs.getString("type").equals("link")){  
  //use hyperlink method to link to another page, ‘url’ stands for another page’s location  
  //the hyperlink is the text  
  out.println("<a  
href='extraview.jsp?CH="+ch+"&ID="+rs.getString("url")+"'></a>");  
  ..... }  
}
```

**Display the option of customer’s ordering, such as ‘buy now’ shown in Figure 5.7:**

```
if(rs.getString("type").equals("order")){  
  out.println("<a  
href='order.jsp?CH="+ch+"&ID="+rs.getString("id")+"'>buy  
now</a>"); }  
if(rs.getInt("enter")==1)out.println("<br/>"); }
```

**Define the function of displaying the customer’s shopping orders in current page, such ‘take the bill’ shown in Figure 5.6:**

```
out.println("client:"+session.getAttribute("name")+ "<br/>");  
.....  
<a href="order.jsp">take the bill</a><br/>  
</p>  
<do type="options" label="home"><go  
href="http://localhost/ExtraWapImage/extraview.jsp?ID=2" /></do>  
<do type="prev" label="back"><prev /></do>  
</card>  
</wml>
```

Since the layout of product catalogs and product lists are exactly similar, as in section 5.1.2 indicated that the methods of editing layouts of catalogs and products are same, I will not repeatedly introduce the programming implementation of product pages. Readers can refer to appendix to find the complete source code. In next section I will introduce the implementation of ordering functionalities.

### 5.2.2 Implementation of Customer ordering

Before I begin to present the implementation of customer ordering functionalities, I should give some background information of **Servlet sessions** which is the kernel method to realize customer ordering functionalities, since I haven't give the introduction of Servlet sessions Chapter 3 Development Tools.

Servlet container provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user. This way is named as 'session'. The servlet container uses HttpSession interface to create a session between users and a Web server which maintain a session in several ways such as using cookies, or rewriting URL.

When a user requests a JSP file that includes session, JSP servlet will send a session identifier, named session ID, to client's navigator. After the user finishes actions this time, navigator will send the session ID to Web server, if this is the first time of client's session request, web server will create a session object for that user, otherwise web server will check the ID of this time and previous ID to identity if they are resemble, and create the correlation between HttpSession and the request of this time.

Sessions' objects are saved in computer's memory, and they depend on navigators' cookies. If a user disables navigators' cookies, session can not be used. And session might be expired in multi-web servers' environment. For instance: Single Web server1 creates the session ID for every session, and this identifier will be saved with session in computer's memory. In other words, if there are more than one Web servers that create their own sessions' identifiers, single Web server1 can not validate its session ID.

In implementation of Customer Interface, I use `setAttribute ()` method to save the data in sessions, `getAttribute ()` to get the data from sessions, and `removeAttribute ()` to erase the records of sessions.

In wapShop customer login simulation page, I use session to save the logging on user information, for example, the name and mobile phone number which will be the specific identifier of customer shopping orders, etc. (Appendix 8.2.7):

```
<%  
    session.setAttribute("name","Mike");  
    session.setAttribute("phone","13641366774");  
    .....  
%>
```

I also use session to save the customer shopping order, and in the end of customer's shopping action. 'Take the bill' function will get the order data from every single

shopping action by using `getAttribute()`, add them together, and save the orders' data in database. For instance, getting order (bill) in `order.jsp` (Appendix 8.2.9):

```

Vector vec = (Vector)session.getAttribute("Bill");
// Vector 'vec' is the shopping cart to save the customer ordering information
    if(vec == null){
        vec = new Vector();
        vec.add(1+" "+text+" "+rs.getString("id"));
// if vec is null, then create a new shopping cart
    }else{
        vec.add((vec.size()+1)+" "+text+" "+rs.getString("id"));
// if the shopping cart has already been existing, then save the new ordering
//information into this shopping cart
    }
    session.setAttribute("Bill",vec);

```

Now we back to Section 5.2.1, when a customer chooses a product, for example 'tent' and he clicks on the hyperlink to tent webpage. He finds the price of tent is acceptable so he decides to buy this product. The simple way to buy this product is to click on 'buy now' and submit his order as following figure indicates, however, this is only a single product shopping. If he wants to buy more products, he should not click on 'submit', back to other product pages and in the homepage click on 'take the bill' to check out all his orders. Furthermore the Customer Interface will give the confirmation of customer orders like red round shown.



Figure 5.8 Screenshot of Customer Ordering

As the WOSS structure introduced in Chapter 4, `saveOrder.jsp` is to calculate the total customer orders and save them into MySQL database. Now I present the implementation of `saveOrder.jsp`.

First, to retrieve the customer's shopping orders by using sessions

```

String userMobile = (String)session.getAttribute("phone");
Vector vec = (Vector)session.getAttribute("Bill");
.....

```

Second, to save customer's shopping orders in userOrder table of database by using SQL, stat.executeUpdate(sql):

```

.....
int total = 0;
if(vec!=null){
    for(int i=(vec.size()-1);i>=0;i--){
        String text = (String)vec.get(i);
        String[] item = text.split("\\;");
        sql = "insert into userorder(userMobile,itemID,itemName,price)
values("+userMobile+", "+item[3]+", "+item[1]+", "+item[2]+")";
        stat.executeUpdate(sql);
        out.println(item[1]+" ");
        out.println("price:"+item[2]+"$ <a
href='orderDel.jsp?setID="+i+"'>del</a><br/>");
        //out.println(text+"<br/>");

// calculate the total price of Customer shopping:
try{
    total += Integer.parseInt(item[2]); bills
    } catch (Exception e) {          }      }

```

All in all, Customer Interface realizes the functionalities defined in WAP shop prototype both in visualization and ordering. Readers currently may understand how I implemented these JSP files to realize the prototype. In next section, I will introduce the implementation of MySQL database.

### 5.3 Implementation MySQL database

In Section 3.3, I presented the brief introduction of using MySQL, and the installation of MySQL database has already been stated, so I will not repeat it.

First, implementation of extrawapimage table, in extrawapimage table, as Section 4.5 indicated, I define 10 table fields.

Field	Type	Null	Key	Default	Extra
id	int(8)		PRI	NULL	auto_increment
type	char(16)	YES		NULL	
text	char(255)	YES		NULL	
align	char(16)	YES		NULL	
url	char(255)	YES		NULL	
enter	int(1)	YES		NULL	
orderid	int(8)	YES		NULL	
pageid	int(8)	YES		NULL	
status	int(1)	YES		NULL	
reserve	char(255)	YES		NULL	

Created by code:

```
CREATE TABLE extrawapimage (  
  id int(8) NOT NULL auto_increment,  
  type char(16) default NULL,  
  text char(255) default NULL,  
  align char(16) default NULL,  
  url char(255) default NULL,  
  enter int(1) default NULL,  
  orderid int(8) default NULL,  
  pageid int(8) default NULL,  
  status int(1) default NULL,  
  reserve char(255) default NULL,  
  PRIMARY KEY (id)  
) TYPE=MyISAM;
```

Second, implementation of products table, in products table, as Section 4.5 indicated, I define 7 table fields.

Field	Type	Null	Key	Default	Extra
id	int(8)		PRI	NULL	auto_increment
name	varchar(16)	YES		NULL	
price	int(8)	YES		NULL	
unit	varchar(16)	YES		NULL	
details	varchar (255)	YES		NULL	
image	varchar (255)	YES		NULL	
reserve	varchar (255)	YES		NULL	

Created by code:

```
CREATE TABLE products (  
  id int(8) NOT NULL auto_increment,  
  name varchar(16) default NULL,  
  price char int(8) default NULL,  
  unit varchar(16) default NULL,  
  details varchar(255) default NULL,  
  image varchar(255) default NULL,  
  reserve varchar(255) default NULL,  
  PRIMARY KEY (id)  
) TYPE=MyISAM;
```

Third, implementation of userorder table, in userorder table, as Section 4.5 indicated, I define 5 table fields.

Field	Type	Null	Key	Default	Extra
id	int(8)		PRI	NULL	auto_increment

userMobile	char (20)	YES		NULL	
itemID	char (16)	YES		NULL	
itemName	char (255)	YES		NULL	
price	int(8)	YES		NULL	

Created by code:

```
CREATE TABLE products (
  id int(8) NOT NULL auto_increment,
  userMobile int(8) default NULL,
  itemID char (16) default NULL,
  itemName char(255) default NULL,
  price int(8) default NULL,
  PRIMARY KEY (id)
) TYPE=MyISAM;
```

Since the method of implementation of users table and showwindows table are almost same as these three tables, I will not repeat them.

After implementation of these tables, I should load the data including product pictures or other data into tables of WOSS system. I will continue the implementation from section 3.3. Under the username 'Mike' with password '123', I adopted 'load' command to load the information required into tables. For instance:

```
>mysql -u Mike -p 123
>use gtom_wap;
> load data local infile "products.txt" into table products;
```

'product.txt' should be the normal text file. It should be written as the sequences of table rows, and divided by key 'TAB'. Since the method of loading other products is exactly same as loading product.txt, I will not repeat them.

## 5.4 Conclusion

In this chapter I presented the implementation of three major modules defined in Chapter 2 and the implementation processes are absolutely following the structure designed in Chapter 4. By using implemented Customer Interface, Administrative Module and MySQL database, WOSS system can realize the prototype defined in Chapter 2 thoroughly.

After system implementation, I will proceed the system test to check whether WOSS system is able to complete visualization functionalities of customer interface and administrative module correctly, and whether it is able to complete ordering functionalities and editing functionalities effectively. In next chapter, I will give the details of the system test.

## 6 System Test

In this Chapter, I present the system test after WOSS system is implemented. The system test is approached from two directions: Component test and Integration test. Since at the beginning of system development, I have already tested the Web server (Orion Application Server) and MySQL database server, when they were installed, I will not discuss the tests of these two systems. Some graphics of the visualization of WOSS system were presented in Chapter 5 by using Opera navigator and Internet Explorer. To avoid repetition in this chapter, here I mainly focus on testing the WOSS system's application by using OpenWave, such as customer shopping applications and administrative module applications.

### 6.1 Component test

The component test is mainly to test the every JSP files of WOSS system, to check whether they are working correctly or not. The visual web pages of WOSS system are divided into Customer Interface and Administrative Module. For Customer Interface, it includes homepage of wapShop (or any other items) listing product catalogs; single product catalog; single product pages, and customer order pages. For Administrative module it includes homepage of WAP SHOP EDIT and Page Editor that includes product catalog editor; product editor; and picture uploading page.

Actually, in Chapter 5 System Implementation, I have already given the visualization of Customer Interface and Administrative Module and they are followed by text description that were stated the functionalities of these pages. Readers can find out the correctness of displaying these interfaces. Those can be considered as the pre-test of Component test.

In the following, to avoid repetition of graphic utilities and functional description, I only discuss the partial test of Customer Interface and Administrative Module briefly, and some figures are quoted from Chapter 5. Similarly, I use Opera navigator and OpenWave navigator simulator of mobile phone (to retrieve technical documents and operation manual, readers please refer to its homepage)<sup>24</sup> to test Customer Interface, and Internet Explorer to test Administrative Module.

#### 6.1.1 Component test of Customer Interface

When a customer logs on wapShop, by typing:  
`http://localhost/ExtraWapImage/wapShop.jsp` (local host can be replaced by 127.0.0.1), he or she can see the product catalogs in Opera navigator. System file, `wapShop.jsp` is responsible for customer identification and `extraview.jsp` is responsible

---

<sup>24</sup> OpenWave: <http://www.openwave.com/us/>, Openwave Systems Inc, 2000-2004

for displaying this item. The visual graphic refers to Figure 5.5. By using OpenWave to test displaying this item and product catalogs that can be displayed on cell phone's navigator, we can see the item from following figure:



Figure 6.1 Component Test of wapShop homepage by using OpenWave

One item should include several different product catalogs, for example, as Figure 6.1 showing, wapShop includes 'sports', 'new DVD' and so on. In addition, on product catalog should include different products. For instance, 'sports' includes products 'tent' and 'water bottle'. This visual page on Opera navigator is referred to Figure 5.6 Sports page. The OpenWave testing page of products is shown as following figure:



Figure 6.2 Component Test of product page by using OpenWave



When a customer wants to buy the products shown on Figure 6.2, the customer interface should show the price of products and feedback information of customer ordering action. 'order.jsp, saveOrder.jsp and ordreDel.jsp' are responsible for customer's ordering functionalities. The visual pages of customer ordering by using Opera navigator can be referred to Figure 5.8. In addition, Testing of the visual pages of shopping by using OpenWave is shown as following figure:



Figure 6.3 Component Test of Customer Ordering by using OpenWave

All in all, after the test of Customer Interface, I concluded that all the \*.jsp files related to Customer Interface are working correctly. Next step is to test the Administrative Module.

### 6.1.2 Component Test of Administrative Module

An administrator logs on WAP SHOP EDIT, by typing `http://localhost/ExtraWapImage/ExtraWapImage.jsp`, 'localhost' can be replaced by 127.0.0.1.

In the component test process of Administrative Module, all the visualization pages, by using Internet Explorer, were listed as the graphics indicated in Chapter 5. ExtraWapImage.jsp is responsible for visualizing WAP SHOP EDIT that is the homepage. PageEdit.jsp is responsible for displaying the Page Editor and Product Editor (the detailed information of the functionalities of JSP files, please refer to Table 4.1 and Table 4.2). For instance, the visualization of homepage WAP SHOP EDIT is referred to Figure 5.1; visualization of Page Editor for product catalogs is

referred to Figure 5.2; visualization of Page Editor for products is referred to Figure 5.3.

I only give the test graphics of uploading picture. When an administrator decides to upload the picture of a product, he or she clicks on 'upload picture' below 'LINK'. The Administrative Module should display the picture uploading webpage, as following figure showing:

Page Editor <a href="#">Back to Homepage</a>									
ID	TYPE	DESCRIPTION	POSITION	LINK	ENTER NEW LINE	ARRANGE SEQUENCE	PAGE NUMBER	STATUS	EDIT DELETE
50	pic		center	<a href="#">UPLOAD PICTURE</a>	<input type="checkbox"/>	1	42	<input type="checkbox"/>	
<a href="#">CREATE NEW CONTENT</a>	PICTURE		CENTER	NUL	<input type="checkbox"/>	1	42	<input type="checkbox"/>	<a href="#">ADD THIS CONTENT</a>

Figure6.4 Test of uploading the picture (1)

Consequently, the navigator display the picture uploading webpage, it is shown as following figure:

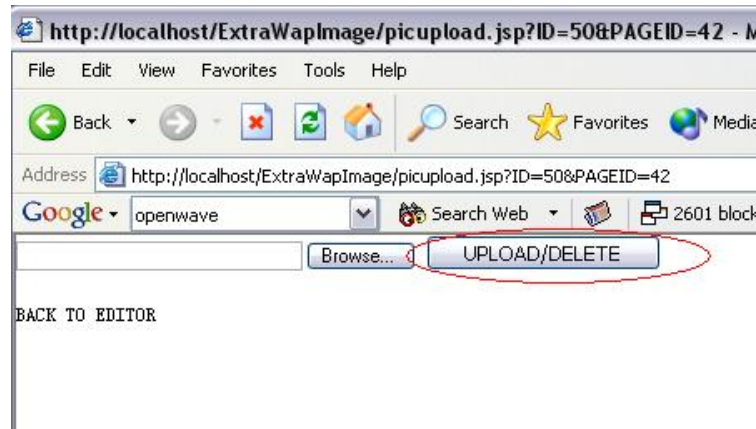


Figure 6.5 Test of uploading pictures (2)

All in all, all the webpages of Administrative Module are displayed correctly. Finally, I finished the Component test of WOSS system. In next section I will discuss the Integration test of WOSS system, including the test of dataflow to MySQL database system.

## 6.2 Integration test

After the component test of WOSS system, I can conclude that all the JSP files display correctly. Then I will do the integration test to find out whether these JSP files work together correctly or not.

Integration test of WOSS system can be divided into three parts, e.g integration test of Administrative editing process, integration test of Customer ordering process, and data saved in MySQL database. In this section, I give the integration test of these three processes by using OpenWave and test of customer order data recorded in database system.

### 6.2.1 Integration test of editing functionalities

When an administrator logs on WAP SHOP EDIT, it shows the existing items in that page. Now I assume Mike as the administrator of WOSS system. He wants to add a new product catalog 'test' into this item. The integration test process of editing functionalities should be as following steps.

Mike firstly inputs 'TEST' below 'DESCRIPTION', chooses the 'TYPE' as 'Link' which means 'TEST' is linked to products edit page that is for adding products and editing their information. Second he ticks on the box status and 'ENTER NEW LINE' on, which means he wants to display 'TEST' in a new row, and finally clicks on 'ADD THIS CATALOG'. It shows:

Page Editor [Back to Homepage](#)









ID	TYPE	DESCRIPTION	POSITION	LINK	ENTER NEW LINE	ARRANGE SEQUENCE	PAGE NUMBER	STATUS	EDIT DELETE
44	pic		center		<input checked="" type="checkbox"/>	4	2	<input checked="" type="checkbox"/>	 
10	link	Sports	center	<a href="#">Sports</a>	<input checked="" type="checkbox"/>	3	2	<input checked="" type="checkbox"/>	 
8	link	new DVD	center	<a href="#">new DVD</a>	<input checked="" type="checkbox"/>	2	2	<input checked="" type="checkbox"/>	 
46	link	TEST	center	<a href="#">TEST</a>	<input checked="" type="checkbox"/>	1	2	<input checked="" type="checkbox"/>	 
6	link	new Phone	center	<a href="#">new Phone</a>	<input checked="" type="checkbox"/>	1	2	<input checked="" type="checkbox"/>	 
36	link	new books	center	<a href="#">new books</a>	<input checked="" type="checkbox"/>	1	2	<input checked="" type="checkbox"/>	 
<a href="#">CREATE NEW PRODUCT CATALOG</a>	PICTURE		CENTER	NUL	<input type="checkbox"/>	1	2	<input type="checkbox"/>	<a href="#">ADD THIS CATALOG</a>

Figure 6.6 Page Editor after adding 'TEST'

So this new catalog is shown in customer shopping page, as following figure shows:



Figure 6.7 Shopping page after adding 'TEST'

But now, catalog 'TEST' is empty, Mike needs to add products to this catalog, so he clicks on 'TEST' below 'LINK' and enter the catalog editing webpage. Mike needs to add 'Girls Picture' to 'TEST' catalog as his new product. So he inputs 'Girls Picture' below 'Description', chooses the type as 'Text', ticks on the box 'status' on, and clicks on 'ADD THIS CONTENT' to display 'Girl's Picture' in 'TEST' shopping page. The Page Editor shows:

-----Page Editor <a href="#">Back to Homepage</a>									
ID	TYPE	DESCRIPTION	POSITION	LINK	ENTER NEW LINE	ARRANGE SEQUENCE	PAGE NUMBER	STATUS	EDIT DELETE
47	text	Girls Picture	center	Girls Picture	<input checked="" type="checkbox"/>	1	46	<input checked="" type="checkbox"/>	
<a href="#">CREATE NEW CONTENT</a>	PICTURE		CENTER	NUL	<input type="checkbox"/>	1	46	<input type="checkbox"/>	<a href="#">ADD THIS CONTENT</a>

Figure 6.8 Page Editor after adding 'Girls Picture'

Obviously only this text description is not enough, so Mike chooses 'PICTURE' below 'TYPE' and uploads the picture of 'Girls Picture'. It shows:

-----Page Editor <a href="#">Back to Homepage</a>									
ID	TYPE	DESCRIPTION	POSITION	LINK	ENTER NEW LINE	ARRANGE SEQUENCE	PAGE NUMBER	STATUS	EDIT DELETE
47	text	Girls Picture	center	Girls Picture	<input checked="" type="checkbox"/>	1	46	<input checked="" type="checkbox"/>	
48	pic		center		<input checked="" type="checkbox"/>	1	46	<input checked="" type="checkbox"/>	
<a href="#">CREATE NEW CONTENT</a>	PICTURE		CENTER	NUL	<input type="checkbox"/>	1	46	<input type="checkbox"/>	<a href="#">ADD THIS CONTENT</a>

Figure 6.9 Page Editor after uploading the picture

In customer TEST shopping page it shows:



Figure 6.10 Shopping page after uploading the picture

The last step is to input the price information for this picture, so Mike chooses 'Order' below 'TYPE', inputs the price information 'Girls Picture; 200' ticks 'ENTER NEW LINE' and 'status' on, and clicks on 'ADD THIS CONTENT'. The page editor shows:

ID	TYPE	DESCRIPTION	POSITION	LINK	ENTER NEW LINE	ARRANGE SEQUENCE	PAGE NUMBER	STATUS	EDIT DELETE
47	text	Girls Picture	center	Girls Picture	<input checked="" type="checkbox"/>	1	46	<input checked="" type="checkbox"/>	
48	pic		center		<input checked="" type="checkbox"/>	1	46	<input checked="" type="checkbox"/>	
49	order	Girls Piture:200	center	Girls Piture;200	<input checked="" type="checkbox"/>	1	46	<input checked="" type="checkbox"/>	
CREATE NEW CONTENT	PICTURE		CENTER	NUL	<input type="checkbox"/>	1	46	<input type="checkbox"/>	ADD THIS CONTENT

Figure 6.11 Page Editor after adding the price

And in this customer shopping page, it display 'buy now' option. If Mike clicks on 'buy now' it will appear the price is 200, as following figure shows:



Figure 6.12 Shopping page after adding the price

From the integration test process of editing functionality and figures indicated above, I can conclude the integration between Administrative module and Customer Interface was correct. All the information edited or added in Page Editor can be correctly reflected in Customer Interface. All in all, the integration test between Customer Interface and Administrative module was successfully finished.

### 6.2.2 Integration test of ordering functionalities and data saved in database

From the above integration test, I can deduce the correctness of integration between Customer Interface and Administrative Module. However, I should still check the ordering functionalities; the data saved in the database and whether the data are recorded correctly or not, for example, the customer orders are saved correctly or not.

The customer order process displayed by Opera navigator is referred from Figure 5.5 to Figure 5.8. Because the correctness of displaying the shopping item, product catalogs and products, I can conclude the part of customer choosing products in the whole ordering functionalities is correct, so I omit the integration test process from customer logging on wapShop to the navigator links to Girl Picture ordering page inherited from previous section. Similarly, the simulated customer is still Mr. Mike. The following figures indicate the correctness of customer ordering confirmation by using OpenWave:

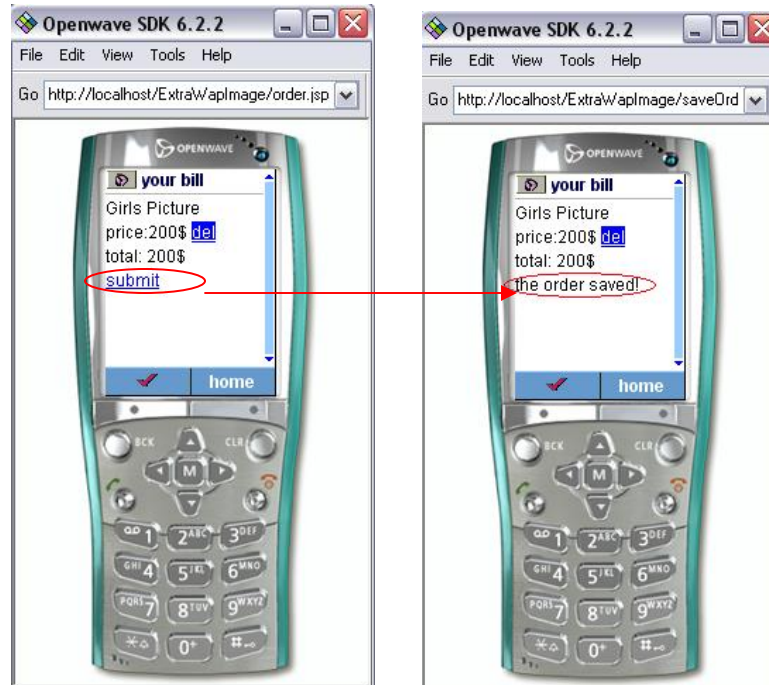


Figure 6.13 Integration test of saving customer's order

Now to retrieve the customer order information, an administrator can query MySQL database system of WOSS system. As Section 2.8 indicated, the functionality of Financial Center is not realized in this thesis; to query the database is the only way for an administrator to retrieve customer order information. The following figure shows the correctness of saving Mike's order in database:

```
mysql> select * from userorder;
+----+-----+-----+-----+-----+
| id | userMobile | itemID | itemName | price |
+----+-----+-----+-----+-----+
| 23 | 13641366774 | 55 | Girls Picture | 200 |
+----+-----+-----+-----+-----+
1 row in set (0.00 sec)
mysql>
```

Figure 6.14 Integration test of saving customer's order in database

All in all, from the graphics indicated above, I conclude that the customer order data are correctly saved in MySQL database management system. In other words, the correctness of integration between Customer Interface and database system is proved.

### 6.3 Conclusion

Based on the results of component test and integration test stated in this chapter and some visualization functionalities pre-adopted in Chapter 5, I can conclude WOSS system works correctly. And it realized the functionalities define in the prototype of

Online Shopping System based on WAP stated in Chapter 2 both in visualization functionalities, ordering functionalities and editing functionalities.

Since WOSS system is a multi-access system, which means it definitely supports remote access to Customer Interface (wapShop, the shopping page) and Administrative Module (WAP SHOP EDID). I used a PC which has IP address: 192.38.79.107 as the Web server, and a laptop which has the IP address: 192.38.79.213 as the client to test the remote access to WOSS system. However, as Section 1.3 indicated, since the user identification system is not realized in WOSS system, readers might raise the problems of security. These points are left to future work.

In the next chapter, I will conclude this whole thesis, present the achievements of this project and suggest the future improvement of WOSS system.



## 7 Conclusion and proposal

In this thesis, I have described the theoretical prototype of Online Shopping System based on WAP, the tools to develop this system, system designs, implementations and tests of WOSS system. In this chapter, I will conclude the achievements of this thesis and present the improvement recommendations of the future work.

### 7.1 Achievements

The objective of this thesis was to realize the prototype of Online Shopping System based WAP by using proper development tools. In the following part, I will discuss what the degree I have succeeded in constructing a feasible and effective Online Shopping System based on WAP.

The work carried out in this thesis can be divided into three major parts:

**The first part** was to present a theoretical foundation of the Online Shopping System based on WAP. In this part, I analyzed the functional requirements of WOSS system in details and presented the prototype of this system which guided the following developing processes.

**The second part** was to present the practical design methods to construct the WOSS system and the proper ways to implement this system. I presented the detailed design information of WOSS system and analyzed the system data structure precisely. In addition, in system implementation, I totally realized the prototype defined in the first part.

**The final part** was to test WOSS system to validate the visualization functionalities and processing functionalities defined are precisely realized and the correctness of database functionalities.

The achievements of this thesis can be summarized as following:

Firstly, I studied the ERP, CRM theories and existing online shopping e-commerce theories and retrieved enough academic and commercial information to define the prototype of WOSS system.

Secondly, I researched the development tools which were adopted to develop some existing online shopping systems and grasped the semantics and syntax of these tools. By following the prototype defined in the first part, I applied these tools in my system and correctly designed and implemented WOSS system.

Finally, by adopting the theories of software engineering, I successfully tested WOSS system and concluded its correctness.

All in all, in the process of development WOSS system, I accumulated the important theories and experiences with developing a complicated management information system and assimilated the most practical and useful suggestions from my supervisor Jens Thyge Kristensen about the prototype definition, system design, system implementation and software engineering theories. These are and will be significantly benefit for my future studies and work.

## **7.2 Future possible improvements of WOSS system**

Presently, the existing WOSS system has perfectly realized the prototype defined in the Chapter 2. In other words, I accomplished the goals written in Section 1.2.2. However, when I implemented WOSS system, I also faced some unexpected problems.

First, the visualization functionalities could be improved in the future, for example, it can be designed more vivid and convenient for customers. It should have the user identification functionalities and so on.

Second, the data fields can be improved in the development of database system of WOSS system, for example, to assign the unique ID for WAP shop data instead of setting the universal id for any data concerned with WOSS system (in Figure 6.6 the ID of pictures links and orders are auto\_increment corresponding to the 'id' defined in MySQL tables which might be confused to readers and users), and to record more detailed information of customer ordering than only recording customer's mobile phone number (in Figure 6.14, when a customer orders a product, the system only records his mobile phone number which is not enough for Financial Center or Logistic Center to process his order, but since Financial Center or Logistic Center to process his order are not realized in this thesis, the functionalities of recording more information of customers will be improved in the future).

Third, the ERP system should consist of more modules mentioned in Section 2.5, for instance, the Logistic system, Financial system and more useful interfaces, and these modules should be developed separately.

Therefore, from the requirements analysis of Online Shopping System based on WAP and development processes of this system, I feel WOSS system should be consummated and improved to be realized as the genuine and multi format ERP system in the future.

## 8 Bibliography and Appendixes

### 8.1 Bibliography

1. J2SDK, <http://java.sun.com/j2se/1.4.2/>, Sun Microsystems, 1994-2004
2. MySQL, [www.mysql.com](http://www.mysql.com), MySQL AB, 1995-2004
3. JDBC, <http://java.sun.com>, Sun Microsystems, 1994-2004
4. Opera Navigator, <http://portal.opera.com/>, Opera Software ASA, 1995-2004
5. Nokia News, [www.nokia.com/search/index.jsp?wsid=8&qt=news](http://www.nokia.com/search/index.jsp?wsid=8&qt=news), Nokia Finland, 2004
6. Organization for Economic Co-operation and Development, [www.oecd.org/home/](http://www.oecd.org/home/)
7. Mobile eBusiness Magic White Paper, [magic-SW.com](http://magic-SW.com), Magic, 2003, USA
8. Practical UML, <http://bdn.borland.com/article/0,1410,31863,00.html#use-case-diagram>
9. CRM: Customer Relationship Management, [www.crm2day.com](http://www.crm2day.com), CRM Today, 2001-2004
10. EJB/J2EE, <http://java.sun.com/products/ejb/>, Sun Microsystems, 1994-2004
11. WAP, [www.w3schools.com](http://www.w3schools.com), W3Schools, Refsnes Data, 1999-2004
12. MySQL reference manual version A4, MySQL AB, 1997-2004
13. JDBC driver of MySQL:  
[http://dev.mysql.com/doc/mysql/en/Java\\_Connector.html](http://dev.mysql.com/doc/mysql/en/Java_Connector.html), MySQL AB, 1997-2004
14. JavaBeans Application specification: <http://java.sun.com/products/javabeans/>, SUN Microsystem, 1994-2004
15. Orion application official site: [www.orionserver.com](http://www.orionserver.com), IronFlere
16. JNDI: <http://java.sun.com/products/jndi/>, SUN Microsystem, 1994-2004
17. DataSource: [www.orionserver.com](http://www.orionserver.com), IronFlere
18. JSP attributes for page description: <http://java.sun.com/products/jsp/docs.html>, SUN Microsystem, 1994-2002
19. Practical statistics, [www.google.com](http://www.google.com), Google Searching, 2004
20. J2SE API documentation: <http://java.sun.com/j2se/1.4.2/docs/api/>, SUN Microsystem, 1994-2004
21. getXXX methods: Methods inherited from interface `java.sql.ResultSet`, [java.sun.com/j2se/1.4.2/docs/api/](http://java.sun.com/j2se/1.4.2/docs/api/). SUN Microsystem, 1994-2004
22. JavaScript Tutorial: [www.w3schools.com](http://www.w3schools.com), Refsnes Data, 1999-2004
23. &nbsp;: [www.sightspecific.com/~mosh/WWW\\_FAQ/nbsp.html](http://www.sightspecific.com/~mosh/WWW_FAQ/nbsp.html), Lone Wolf, 1993-2003
24. OpenWave: <http://www.openwave.com/us/>, Openwave Systems Inc, 2000-2004

## 8.2 Source code

### 8.2.1 ExtraWapImage.jsp

```
<% @ page language="java"%>
<% @ page import="java.sql.*" %>
<% @ page import="com.gtom.wap.*" %>
<% @ page session="true" %>
<%
request.setCharacterEncoding ("GBK");
String ch=request.getParameter ("CH");
String id=request.getParameter ("ID");
String htmltitle="WAP SHOP EDIT";
String PAGES=request.getParameter ("PAGES");
int pages=(PAGES!=null)?Integer.parseInt(PAGES):0;

String sql=null;
ResultSet rs=null;

Connection Conn = ConnLoader.getInstance ().getConnection (this);
Statement stat = Conn.createStatement ();

if(pages==0)
sql="select * from ExtraWapImage where type='top' order by orderid desc";
else sql="select * from ExtraWapImage where pageid="+pages+" order by orderid desc";
rs=stat.executeQuery( sql );
//if(rs.next()){
%>
<% @ include file="include/htmlhead.inc" %>
<script language=javascript>
<!--
function changechecked(checkbut){
var v =checkbut.checked;
if (v) {
checkbut.checked = true;
}
}
function quickmodify(id){
var f = document.forms["sort"];
var v =id;
document.forms["sort"].elements["id"].value=v;
document.forms["sort"].elements["type"].value=document.forms["sort"].elements["type"+id].value;
document.forms["sort"].elements["text"].value=document.forms["sort"].elements["text"+id].value;
document.forms["sort"].elements["align"].value=document.forms["sort"].elements["align"+id].value;
document.forms["sort"].elements["url"].value=document.forms["sort"].elements["url"+id].value;
if (document.forms["sort"].elements["enter"+id].checked)
document.forms["sort"].elements["enter"].value=1;
else
```

```

document.forms["sort"].elements["enter"].value=0;
document.forms["sort"].elements["orderid"].value=document.forms["sort"].elements["orderid"+id].value;
if (document.forms["sort"].elements["status"+id].checked)
document.forms["sort"].elements["status"].value=1;
else
document.forms["sort"].elements["status"].value=0;
var surl =
"ewiupdate.jsp?ID="+v+"&TYPE="+document.forms["sort"].elements["type"].value+"&TE
XT="+document.forms["sort"].elements["text"].value+"&ALIGN="+document.forms["sort"]
.elements["align"].value+"&URL="+document.forms["sort"].elements["url"].value+"&ENTE
R="+document.forms["sort"].elements["enter"].value+"&ORDERID="+document.forms["sor
t"].elements["orderid"].value+"&STATUS="+document.forms["sort"].elements["status"].val
ue;
window.open(surl,"_self");
//document.forms["sort"].submit();
}
function selectall(v){
var f = document.forms["sort"];
for (i=0; i<f.elements.length;i++)
if (f.elements[i].value=="on") f.elements[i].checked = v;
}
-->
</script>
<BODY>
<form name="sort" method="post" action="ewiupdate.jsp">
<input type="hidden" name=id>
<input type="hidden" name=type value="top">
<input type="hidden" name=text>
<input type="hidden" name=align>
<input type="hidden" name=url>
<input type="hidden" name=enter>
<input type="hidden" name=orderid>
<input type="hidden" name=status value=0>
<table border="1" cellspacing="0" cellpadding="0" width="100%">
<tr>
<td height="42" colspan="10" valign="middle">
<div align="left"><font color="#5f8ac5">
<font>WAP SHOP EDIT</font>
</div>
</td>
</tr>
<tr>
<td height="27" align="center"><font color="#999999">ID</font></td>
<td height="27" align="center"><font color="#999999">TYPE</font></td>
<td height="27" align="center"><font color="#999999">TITLE</font></td>
<td height="27" align="center"><font color="#999999">POSITION</font></td>
<td height="27" align="center"><font color="#999999">EDIT</font></td>
<td height="27" align="center"><font color="#999999">ENTER NEW LINE</font></td>
<td height="27" align="center"><font color="#999999">ARRANGE SEQUENCE</font></td>
<td height="27" align="center"><font color="#999999">PAGE NUMBER</font></td>
<td height="27" align="center"><font color="#999999">STATUS</font></td>

```

```

<td height="27" align="center"><font color="#999999"><nobr>EDIT DELETE</nobr></font></td>
</tr>
<%
while(rs.next()){
%>
<tr>
<td align="center"><%=rs.getString("id")%></td>
<td align="center"><%=rs.getString("type")%><input type=hidden name=type<%=rs.getString("id")%>
value="<%=rs.getString("type")%>"></td>
<td align="center">
<input type=text name=text<%=rs.getString("id")%> value="<%=rs.getString("text")%>">
</td>
<td align="center">NUL<input type=hidden name=align<%=rs.getString("id")%>
value="NUL"></td>
<td align="center"><a
href="PageEdit.jsp?PAGEID=<%=rs.getString("url")%>">EDIT</a><input type=hidden
name=url<%=rs.getString("id")%> value="<%=rs.getString("url")%>"></td>
<td align="center">NUL<input type=hidden name=enter<%=rs.getString("id")%> value="NUL"></td>
<td align="center">
<select name="orderid<%=rs.getInt("id")%>">
<option value="<%=rs.getString("orderid")%>" selected><%=rs.getString("orderid")%>
</option>
<%
for(int i = 1; i < 10 ;i++)
{
out.println("<option value=\" + i + "\"> + i + "</option>");
}
%>
</select></td>
<td align="center"><%=rs.getString("pageid")%></td>
<td align="center">
<input type="checkbox" name="status<%=rs.getString("id")%>" onclick="changechecked(this)"
<%
if(rs.getInt("status") == 1){
out.print("value='on'");
out.print("checked");
}
%>>
</td>
<td align="center">
<a href = "#" onclick="quickmodify (<%=rs.getInt("id")%>)"></a>&nbsp;
<a href = "ewidel.jsp?ID=<%=rs.getString("id")%>&FROM=EWI"></a>
</td>
</tr>
<%
} // while
%>

```

```

<tr>
<td align="center">CREATE NEW</td>
<td align="center">HOMEPAGE<input type=hidden name=type0 value="top"></td>
<td align="center">
<input type=text name=text0>
</td>
<td align="center">NUL<input type=hidden name=align0 value="nul"></td>
<td align="center">NUL<input type=hidden name=url0 value="0"></td>
<td align="center">NUL<input type=hidden name=enter0 value="nul"></td>
<td align="center">
<select name="orderid0">
<%
for(int i = 1; i < 300 ;i++)
{
out.println("<option value=\"\" + i + \"\">\" + i + \"</option>");
}
%>
</select>
</td>
<td align="center">Automatically Distribute</td>
<td align="center">
<input type="checkbox" name="status0" onclick="changechecked (this)">
</td>
<td align="center">
<a href = "#" onclick="quickmodify (0)">Add this Title</a>
</td>
</tr>
</table>
</form>
</BODY>
</HTML>
<%
//}
if(rs!=null)rs.close();
//conn.freeConnection();
Conn.close();
%>

```

### 8.2.2 ewiupdate.jsp

```

<% @ page import="java.sql.*" %>
<html>
<head>
<title></title>
<SCRIPT LANGUAGE="JavaScript">
<!--
if (navigator.userAgent.indexOf("MSIE 5") != -1)
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/ows.css\" TYPE=\"text/css\">");

```

```

document.write("<STYLE>");
document.write("<<!-->");
document.write("<A: hover {color:386BCC}>");
document.write("</-->");
document.write("</STYLE>");
}
else
{
if (navigator.appName == "Netscape")
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/owsns.css\" TYPE=\"text/css\">");
}
else
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/owsns.css\" TYPE=\"text/css\">");
}
}
}
//-->
</SCRIPT>
<SCRIPT language=JavaScript src="libraries/expcolla.js"></SCRIPT>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#5f8ac5" text="#000000" leftmargin="0" marginwidth="0">
<%
String id=request.getParameter("ID");
String type=request.getParameter("TYPE");
String text=request.getParameter("TEXT");
String align=request.getParameter("ALIGN");
String url=request.getParameter("URL");
String enter=request.getParameter("ENTER");
String orderid=request.getParameter("ORDERID");
String pageid=request.getParameter("PAGEID");
String status=request.getParameter("STATUS");
String sql="";

Connection Conn = com.gtom.wap.ConnLoader.getInstance().getConnection(this);
java.sql.Statement stat = Conn.createStatement();

if(!id.equals("0")){
sql="UPDATE ExtraWapImage SET
type="+type+",text="+text+",align="+align+",url="+url+",enter="+enter+",orderid="+or
derid+",status="+status+" WHERE ID="+id;
stat.executeUpdate(sql);
}else{
sql="insert into ExtraWapImage(type,text,align,enter,orderid,status)
values('top','"+text+"','nul','nul','"+orderid+"','"+status+")";
stat.executeUpdate(sql);
sql="select * from ExtraWapImage where type='top' and text='"+text+"' and
orderid="+orderid;
ResultSet rs=stat.executeQuery(sql);
rs.next();

```



```

sql="UPDATE ExtraWapImage SET url='"+rs.getString("id")+ "',pageid=0 where
id='"+rs.getString("id");
stat.executeUpdate(sql);
}
out.println(sql);
stat.close();
Conn.close();
response.sendRedirect("ExtraWapImage.jsp");
%>
</body>
</html>

```

### 8.2.3 PageEdit.jsp

```

<% @ page language="java"%>
<% @ page import="java.sql.*" %>
<% @ page import="com.gtom.wap.*" %>
<% @ page session="true" %>
<%
request.setCharacterEncoding("GBK");
String ch=request.getParameter("CH");
String id=request.getParameter("ID");
String pt=request.getParameter("PT");
String htmltitle="-----Page Editor";
String PAGEID=request.getParameter("PAGEID");
int pageid=(PAGEID!=null)?Integer.parseInt(PAGEID):0;
if(pageid==0){
out.println("<a href='ExtraWapImage.jsp'>Enter new title and Edit later</a>");
}else{
String sql=null;
ResultSet rs=null;

Connection Conn = ConnLoader.getInstance().getConnection(this);
Statement stat = Conn.createStatement() ;

sql="select * from ExtraWapImage where pageid="+pageid+" order by orderid desc";
rs=stat.executeQuery( sql );
//if(rs.next()){
%>
<% @ include file="include/htmlhead.inc" %>
<script language=javascript>
<!--
function changechecked(checkbut){
var v =checkbut.checked;
if (v) {
checkbut.checked = true;
}
}
function quickmodify(id,opt){

```

```

var f = document.forms["sort"];
var v =id;
document.forms["sort"].elements["id"].value=v;
document.forms["sort"].elements["type"].value=document.forms["sort"].elements["type"+id].value;
document.forms["sort"].elements["text"].value=document.forms["sort"].elements["text"+id].value;
document.forms["sort"].elements["align"].value=document.forms["sort"].elements["align"+id].value;
document.forms["sort"].elements["url"].value=document.forms["sort"].elements["url"+id].value;
document.forms["sort"].elements["pageid"].value=document.forms["sort"].elements["pageid"+id].value;
if (document.forms["sort"].elements["enter"+id].checked)
document.forms["sort"].elements["enter"].value=1;
else
document.forms["sort"].elements["enter"].value=0;
document.forms["sort"].elements["orderid"].value=document.forms["sort"].elements["orderid"+id].value;
if (document.forms["sort"].elements["status"+id].checked)
document.forms["sort"].elements["status"].value=1;
else
document.forms["sort"].elements["status"].value=0;
var surl =
"ewipageupdate.jsp?ID="+v+"&TYPE="+document.forms["sort"].elements["type"].value+"
&TEXT="+document.forms["sort"].elements["text"].value+"&ALIGN="+document.forms["s
ort"].elements["align"].value+"&URL="+document.forms["sort"].elements["url"].value+"&E
NTER="+document.forms["sort"].elements["enter"].value+"&PAGEID="+document.forms["
sort"].elements["pageid"].value+"&ORDERID="+document.forms["sort"].elements["orderid
"].value+"&STATUS="+document.forms["sort"].elements["status"].value+"&OPT="+opt;
window.open(surl,"_self");
//document.forms["sort"].submit();
}
function selectall(v){
var f = document.forms["sort"];
for (i=0;i<f.elements.length;i++)
if (f.elements[i].value=="on") f.elements[i].checked = v;
}
-->
</script>
<BODY>
<form name="sort" method="post" action="ewipageupdate.jsp">
<input type=hidden name=id>
<input type=hidden name=type>
<input type=hidden name=text>
<input type=hidden name=align>
<input type=hidden name=url>
<input type=hidden name=enter>
<input type=hidden name=orderid>
<input type=hidden name=pageid>
<input type=hidden name=status value=0>
<table border="1" cellspacing="0" cellpadding="0" width="100%">
<tr>
<td height="42" colspan="10" valign="middle">
<div align="left"><font color="#5f8ac5">
<font>-----Page Editor</font>
<a href='ExtraWapImage.jsp'>Back to Homepage</a>

```

```

</div>
</td>
</tr>
<tr>
<td height="27" align="center"><font color="#999999">ID</font></td>
<td height="27" align="center"><font color="#999999">TYPE</font></td>
<td height="27" align="center"><font color="#999999">DESCRIPTION</font></td>
<td height="27" align="center"><font color="#999999">POSITION</font></td>
<td height="27" align="center"><font color="#999999">LINK</font></td>
<td height="27" align="center"><font color="#999999">ENTER NEW LINE</font></td>
<td height="27" align="center"><font color="#999999">ARRANGE SEQUENCE</font></td>
<td height="27" align="center"><font color="#999999">PAGE NUMBER</font></td>
<td height="27" align="center"><font color="#999999">STATUS</font></td>
<td height="27" align="center"><font color="#999999"><nobr>EDIT DELETE</nobr></font></td>
</tr>
<%
while(rs.next()){
%>
<tr>
<td align="center"><%=rs.getString("id")%></td>
<td align="center"><%=rs.getString("type")%><input type=hidden
name=type<%=rs.getString("id")%> value="<%=rs.getString("type")%>"></td>
<td align="center">
<input type=text name=text<%=rs.getString("id")%> value="<%=rs.getString("text")%>">
</td>
<td align="center">
<select name="align<%=rs.getString("id")%>">
<option value="<%=rs.getString("align")%>" selected><%=rs.getString("align")%>
</option>
<option value="center">CENTER</option>
<option value="left">LEFT</option>
<option value="right">RIGHT</option>
</select>
</td>
<td align="<%=rs.getString("align")%>">
<%
if(rs.getString("type").equals("order"))out.println("<h4>"+rs.getString("text")+<h4>");
if(rs.getString("type").equals("link"))out.println("<a
href=\"PageEdit.jsp?PAGEID="+rs.getString("url")+\">"+(((rs.getString("text")!=null)&&(!
rs.getString("text").equals("")))?rs.getString("text"):"EDIT")+</a><input type=hidden
name=url"+rs.getString("id")+\" value=\""+rs.getString("url")+\">");
if(rs.getString("type").equals("pic")){
String ss="UPLOAD PICTURE";
if(rs.getString("url")!=null&&rs.getString("url").startsWith("/wap/"))ss="<img
src=\""+rs.getString("url")+\">";
%>
<a href="picupload.jsp?ID=<%=rs.getString("id")%>&PAGEID=<%=pageid%>"><%=ss%></a>
<input type=hidden name=url<%=rs.getString("id")%> value="<%=rs.getString("url")%>">
<%
}
if(rs.getString("type").equals("text"))out.println((((rs.getString("text")!=null)&&(!rs.getStrin

```

```

g("text").equals("")))?rs.getString("text"):"You haven't input Text!")+"<input type=hidden
name=url"+rs.getString("id")+ " value=\"\">";
%>
</td>
<td align="center"><input type="checkbox" name="enter"<%=rs.getString("id")%>"
onclick="changechecked(this)"
<%
if(rs.getInt("enter") == 1){
out.print("value='on'");
out.print("checked");
}
%>>
</td>
<td align="center">
<select name="orderid"<%=rs.getInt("id")%>">
<option value="<%=rs.getString("orderid")%>" selected><%=rs.getString("orderid")%>
</option>
<%
for(int i = 1; i < 300 ;i++)
{
out.println("<option value=\"\" + i + \"\">" + i + "</option>");
}
%>
</select>
</td>
<td align="center"><%=pageid%><input type=hidden name=pageid<%=rs.getString("id")%>
value="<%=pageid%>"></td>
<td align="center">
<input type="checkbox" name="status"<%=rs.getString("id")%>" onclick="changechecked(this)"
<%
if(rs.getInt("status") == 1){
out.print("value='on'");
out.print("checked");
}
%>>
</td>
<td align="center">
<a href = "#" onclick="quickmodify (<%=rs.getInt("id")%>,'1')"></a>&nbsp;
<a href = "ewidel.jsp?ID=<%=rs.getString("id")%>&PAGEID=<%=pageid%>"></a>
</td>
</tr>
<%
} // while
%>
<tr>
<td align="center">NEW CONTENT</td>
<td align="center">

```

```

<select name="type<%=pageid%>">
<option value="pic">PICTURE</option>
<option value="text">TEXT</option>
<option value="link">LINK</option>
<option value="order">ORDER</option>
</select>
</td>
<td align="center">
<input type="text" name="text<%=pageid%>">
</td>
<td align="center">
<select name="align<%=pageid%>">
<option value="center">CENTER</option>
<option value="left">LEFT</option>
<option value="right">RIGHT</option>
</select>
</td>
<td align="center">NUL<input type="hidden" name="url<%=pageid%>" value="0"></td>
<td align="center">
<input type="checkbox" name="enter<%=pageid%>" onclick="changechecked(this)">
</td>
<td align="center">
<select name="orderid<%=pageid%>">
<%
for(int i = 1; i < 300 ;i++)
{
out.println("<option value=\" + i + \">\" + i + "</option>");
}
%>
</select>
</td>
<td align="center"><%=PAGEID %><input type="hidden" name="pageid<%=pageid%>"
value="<%=pageid%>"></td>
<td align="center">
<input type="checkbox" name="status<%=pageid %>" onclick="changechecked(this)">
</td>
<td align="center">
<a href = "#" onclick="quickmodify (<%=pageid %>,'0')">ADD NEW CONTENT</a>
</td>
</tr>
</table>
</form>
</BODY>
</HTML>
<%
//}
if(rs!=null)rs.close();
//conn.freeConnection();
Conn.close();
} //if(pageid==0);else {}
%>

```

## 8.2.4 ewidel.jsp

```
<% @ page import="java.sql.*" %>
<html>
<head>
<title></title>
<SCRIPT LANGUAGE="JavaScript">
<!--
if (navigator.userAgent.indexOf("MSIE 5") != -1)
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/ows.css\" TYPE=\"text/css\">");
document.write("<STYLE>");
document.write("<<!-->");
document.write("<A: hover {color:386BCC}>");
document.write("</-->");
document.write("</STYLE>");
}
else
{
if (navigator.appName == "Netscape")
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/owsns.css\" TYPE=\"text/css\">");
}
else
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/owsns.css\" TYPE=\"text/css\">");
}
}
}
//-->
</SCRIPT>
<SCRIPT language=JavaScript src="libraries/expcolla.js"></SCRIPT>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#5f8ac5" text="#000000" leftmargin="0" marginwidth="0">
<%
String id=request.getParameter("ID");
String type=request.getParameter("TYPE");
String text=request.getParameter("TEXT");
String align=request.getParameter("ALIGN");
String url=request.getParameter("URL");
String enter=request.getParameter("ENTER");
String orderid=request.getParameter("ORDERID");
String pageid=request.getParameter("PAGEID");
String status=request.getParameter("STATUS");
String sql="";

if(id!=null){
Connection Conn = com.gtom.wap.ConnLoader.getInstance().getConnection(this);
java.sql.Statement stat = Conn.createStatement();
sql="delete from ExtraWapImage where id="+id;
```

```

stat.executeUpdate(sql);
out.println(sql);
stat.close();
Conn.close();
}
String returnUrl="PageEdit.jsp?PAGEID="+pageid;
if(request.getParameter("FROM")!=null&&request.getParameter("FROM").equals("EWI"))
returnUrl="ExtraWapImage.jsp";
response.sendRedirect(returnUrl);
%>
</body>
</html>

```

### 8.2.5 ewipageupdate.jsp

```

<% @ page import="java.sql.*" %>
<html>
<head>
<title></title>
<SCRIPT LANGUAGE="JavaScript">
<!--
if (navigator.userAgent.indexOf("MSIE 5") != -1)
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/ows.css\" TYPE=\"text/css\">");
document.write("<STYLE>");
document.write("<<!-->");
document.write("<A: hover {color:386BCC}>");
document.write("</-->");
document.write("</STYLE>");
}
else
{
if (navigator.appName == "Netscape")
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/owsns.css\" TYPE=\"text/css\">");
}
else
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/owsns.css\" TYPE=\"text/css\">");
}
}
//-->
</SCRIPT>
<SCRIPT language=JavaScript src="libraries/expcolla.js"></SCRIPT>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#5f8ac5" text="#000000" leftmargin="0" marginwidth="0">
<%
String id=request.getParameter("ID");

```

```

String type=request.getParameter("TYPE");
String text=request.getParameter("TEXT");
String align=request.getParameter("ALIGN");
String url=request.getParameter("URL");
String enter=request.getParameter("ENTER");
String orderid=request.getParameter("ORDERID");
String pageid=request.getParameter("PAGEID");
String status=request.getParameter("STATUS");
String opt=request.getParameter("OPT");
String sql="";

Connection Conn = com.gtom.wap.ConnLoader.getInstance().getConnection(this);
java.sql.Statement stat = Conn.createStatement();

if(!opt.equals("0")){
sql="UPDATE ExtraWapImage SET
type="+type+",text="+text+",align="+align+",url="+url+",enter="+enter+",orderid="+or
derid+",status="+status+" WHERE ID="+id;
stat.executeUpdate(sql);
}else{
sql="insert into ExtraWapImage(type,text,align,enter,orderid,pageid,status)
values("+type+", "+text+", "+align+", "+enter+", "+orderid+", "+pageid+", "+status+)";
stat.executeUpdate(sql);
if(type.equals("link")){
sql="select * from ExtraWapImage where pageid="+pageid+" and type='link'";
ResultSet rs=stat.executeQuery(sql);
while(rs.next()){
sql="UPDATE ExtraWapImage SET url="+rs.getString("id")+" where id="+rs.getString("id");
stat.executeUpdate(sql);
}
}
}
out.println(sql);
stat.close();
Conn.close();
response.sendRedirect("PageEdit.jsp?PAGEID="+pageid);
%>
</body>
</html>

```

### 8.2.6 picupload.jsp

```

<% @ page import="com.jspsmart.upload.*"%>
<% @ page import="java.sql.*" %>
<%
String id=request.getParameter("ID");
String pageid=request.getParameter("PAGEID");
%>
<html>

```



```

<head>
<title></title>
<SCRIPT LANGUAGE="JavaScript">
<!--
if (navigator.userAgent.indexOf("MSIE 5") != -1)
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/ows.css\" TYPE=\"text/css\">");
document.write("<STYLE>");
document.write("<<!-->");
document.write("<A:hover {color:386BCC}>");
document.write("</-->");
document.write("</STYLE>");
}
else
{
if (navigator.appName == "Netscape")
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/owsns.css\" TYPE=\"text/css\">");
}
else
{
document.write("<LINK REL=STYLESHEET HREF=\"libraries/owsns.css\" TYPE=\"text/css\">");
}
}
//-->
</SCRIPT>
<SCRIPT language=JavaScript src="libraries/expcolla.js"></SCRIPT>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<jsp:useBean id="myUpload" scope="page" class="com.jspsmart.upload.SmartUpload" />
<jsp:setProperty name="myUpload" property="*" />
<style type="text/css">
A:link,A:active,A:visited{TEXT-DECORATION:none ;Color:#000000}
A:hover{TEXT-DECORATION: underline;Color:#4455aa}

BODY{
FONT-SIZE: 12px;
COLOR: #000000;
FONT-FAMILY: xxx;
background-color: #FFFFFF;
scrollbar-face-color: #DEE3E7;
scrollbar-highlight-color: #FFFFFF;
scrollbar-shadow-color: #DEE3E7;
scrollbar-3dlight-color: #D1D7DC;
scrollbar-arrow-color: #006699;
scrollbar-track-color: #EFEFEF;
scrollbar-darkshadow-color: #98AAB1;
}
TD{
font-family: xxx;
font-size: 12px;

```

```

line-height: 15px;
}
th
{
background-color: #4455aa;
color: white;
font-size: 12px;
font-weight:bold;
}
td.TableTitle2
{
background-color: #E4E8EF;
}
td.TableBody1
{
background-color: #FFFFFF;
}
td.TableBody2
{
background-color: #E4E8EF;
}
td.TopLighNav2
{
background-color:#FFFFFF
}
.tableBorder1
{
width:97%;
border: 1px;
background-color: #6595D6;
}
.tableBorder2
{
width:97%;
border: 1px #DEDEDE solid;
background-color: #EFEFEF;
}

#TableTitleLink A:link, #TableTitleLink A:visited, #TableTitleLink A:active {
COLOR: #FFFFFF;
TEXT-DECORATION: none;
}
#TableTitleLink A:hover {
COLOR: #FFFFFF;
TEXT-DECORATION: underline;}

input,select,Textarea{
font-family:Tahoma,Verdana,xxx; font-size: 12px; line-height: 15px;}
}
.normalTextSmall
{

```

```

font-size : 11px;
color : #000000;
font-family: Verdana, Arial, Helvetica, sans-serif;
}

</style>
</head>
<body bgcolor=background-color: #FFFFFF; text="#000000" leftmargin="0"
topmargin="0">
<table border="0" cellspacing="0" cellpadding="0" width=100%>
<tr>
<td class=tbody1>
<%
String status = "";
int size = 30;
String filename = request.getParameter("filename");
String act = request.getParameter("act");
boolean upok = false;
if ((act!=null)&&(act.equals("upload"))){
upok =true;
myUpload.initialize(pageContext);
try{
myUpload.upload();
com.jspmart.upload.File myFile = myUpload.GetFiles().getFile(0);
String tail = myFile.getFileExt();
if ((tail==null)||(tail.equals(""))){
throw new NegativeArraySizeException();
}
else if (myFile.getSize() > 8192){
upok = false;
status = "FILE TOO LARGE > 8K";
size = 18;
}
else if (!tail.toLowerCase().equals("png")){
upok = false;
status = "ONLY png FORMAT";
size = 18;
}
else if ((filename==null)||(filename.equals(""))){
upok = false;
status = "REFRESH AND UPLOAD AGAIN";
size = 18;
}
else{
upok = false;
size = 24;
String filepath = "/wap/images/uploadpic/"+filename+".png";
myFile.saveAs(filepath, 1);
//out.println("<img id=face src='"+filepath+" width=30 height=30>");
//out.println("<SCRIPT>parent.document.forms[0].content.value = '<img
src='"+filepath+"\" alt=\"\" /><br/>'+parent.document.forms[0].content.value</SCRIPT>");

```

```

out.println("<SCRIPT>parent.document.images['conpic'].src='"+filepath+"';parent.document.
forms[0].conpicurl.value='http://211.94.188.42/'+filepath+'</SCRIPT>");
status = "SUCCESSFUL UPLOAD";

```

```

String sql="";
Connection Conn = com.gtom.wap.ConnLoader.getInstance().getConnection(this);
Statement stat = Conn.createStatement();
sql="update ExtraWapImage set url='"+filepath+"' where id="+id;
stat.executeUpdate(sql);
out.println(sql);
stat.close();
Conn.close();
}
}
catch (NegativeArraySizeException nae){
upok = false;
String upic = "/wap/images/uploadpic/"+filename+".png";
java.io.File picfile = new java.io.File(pageContext.getServletContext().getRealPath(upic));
if (picfile.exists()){
if(picfile.delete()){
out.println("<SCRIPT>parent.document.images['conpic'].src='/wap/images/uploadpic/nopic.g
if';parent.document.forms[0].conpicurl.value='</SCRIPT>");
status = "SUCCESSFUL DELETE";
size = 24;
}else{
status = "PICTURE NOT EXIST";
size = 24;
}
}
}
}
}
if (!upok){
%>
<form name="form" method="post"
action="picupload.jsp?ID=<%=id%>&PAGEID=<%=pageid%>&act=upload&filename=<%=
pageid+"_" +id%>" enctype="multipart/form-data" >
<input type="file" name="file" size="<%=size%>">
<input type="submit" name="submit1" value="UPLOAD/DELETE"
onclick="document.form.submit1.disabled=true,document.form.submit()">
<font color="red"><%=status%></font>
</form>
<a href="PageEdit.jsp?PAGEID=<%=pageid%>">BACK TO EDITOR</a><br>
<%
}
%>
</td>
</tr>
</table>
</body>
</html>

```

## 8.2.7 wapShop.jsp

```
<% @ page language="java" contentType="text/vnd.wap.wml;charset=UTF-8" %>
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<head>
<meta http-equiv="Cache-Control" content="max-age=0" />
<meta http-equiv="Cache-control" content="no-cache" />
</head>
<% @ page import="java.sql.*" %>
<% @ page import="com.gtom.wap.*" %>
<card id='welcome' title='welcome'>
<p>
welcome
</p>
<%
session.setAttribute("name","Mike");
session.setAttribute("phone","13641366774");
session.setAttribute("address","2nd east street 1# building 204.");
response.sendRedirect("extraview.jsp");
%>
<do type="options" label="home"><go href="http://localhost/ExtraWapImage/extraview.jsp"
/></do>
<do type="prev" label="back"><prev /></do>
</card>
</wml>
```

## 8.2.8 extraview.jsp

```
<% @ page language="java" contentType="text/vnd.wap.wml;charset=UTF-8" %>
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<head>
<meta http-equiv="Cache-Control" content="max-age=0" />
<meta http-equiv="Cache-control" content="no-cache" />
</head>
<% @ page import="java.sql.*" %>
<% @ page import="com.gtom.wap.*" %>
<%
String ch    = request.getParameter("CH");
String id    = request.getParameter("ID");
if(id==null) id = "2";
Connection Conn = ConnLoader.getInstance().getConnection(this);
Statement stat = Conn.createStatement();
String sql    = "select * from ExtraWapImage where id="+id;
```

```

ResultSet rs = stat.executeQuery(sql);
if(rs.next())out.println("<card id='welcome'
title='"+rs.getString("text").trim()+">");//util.gb2u(rs.getString("text").trim()+">");
else out.println("<card id='welcome' title='&#x7279;&#x522b;&#x7b56;&#x5212;>");
boolean p = false;
boolean pp = false;
sql = "select * from ExtraWapImage where pageid="+id+" and status=1 order by orderid desc";
rs = stat.executeQuery(sql);
String align=null;
while(rs.next()){
if(align==null||!align.equals(rs.getString("align")))p=true;
if(p){
if(pp)out.println("</p>");
out.println("<p align='"+rs.getString("align")+">");
p=false;
pp=true;
}
align=rs.getString("align");
if(rs.getString("type").equals("pic")){
out.println("<img src='"+rs.getString("url")+"" alt='"+rs.getString("text").trim()+">");
}
if(rs.getString("type").equals("text")){
out.println(rs.getString("text").trim());
}
if(rs.getString("type").equals("link")){
out.println("<a
href='extraview.jsp?CH="+ch+"&ID="+rs.getString("url")+"">"+rs.getString("text").trim()+"</a
>");//util.gb2u(rs.getString("text").trim()+"</a>");
}
if(rs.getString("type").equals("order")){
out.println("<a href='order.jsp?CH="+ch+"&ID="+rs.getString("id")+"">buy
now</a>");
}
if(rs.getInt("enter")==1)out.println("<br/>");
}
rs.close();
Conn.close();
out.println("<br/>");
out.println("client:"+session.getAttribute("name")+ "<br/>");
//out.println("phone:"+session.getAttribute("phone")+ "<br/>");
//out.println("address:"+session.getAttribute("address")+ "<br/>");
%>
<a href="order.jsp">take the bill</a><br/>
</p>
<do type="options" label="home"><go
href="http://localhost/ExtraWapImage/extraview.jsp?ID=2" /></do>
<do type="prev" label="back"><prev /></do>
</card>
</wml>

```

## 8.2.9 order.jsp

```
<% @ page language="java" contentType="text/vnd.wap.wml;charset=UTF-8" %>
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<head>
<meta http-equiv="Cache-Control" content="max-age=0" />
<meta http-equiv="Cache-control" content="no-cache" />
</head>
<% @ page import="java.sql.*" %>
<% @ page import="java.util.*" %>
<% @ page import="com.gtom.wap.*" %>
<card id='order' title='your bill'>
<p>
<%
String id = request.getParameter("ID");
Connection Conn = ConnLoader.getInstance().getConnection(this);
Statement stat = Conn.createStatement();
String sql = null;
ResultSet rs = null;
if(id!=null){
sql = "select * from ExtraWapImage where id="+id;
rs = stat.executeQuery(sql);
if(rs.next()){
String text = rs.getString("text").trim();
Vector vec = (Vector)session.getAttribute("Bill");
if(vec == null){
vec = new Vector();
vec.add(1+";"+text+";"+rs.getString("id"));
}else{
vec.add((vec.size()+1)+";"+text+";"+rs.getString("id"));
}
session.setAttribute("Bill",vec);
}
}
Vector vec = (Vector)session.getAttribute("Bill");
int total = 0;
if(vec!=null){
for(int i=(vec.size()-1);i>=0;i--){
String text = (String)vec.get(i);
String[] item = text.split("\\;");
out.println(item[1]+" ");
out.println("price:"+item[2]+"$ <a href='orderDel.jsp?setID="+i+"'>del</a><br/>");
//out.println(text+"<br/>");
}
try{
total += Integer.parseInt(item[2]);
}catch(Exception e){
}
}
```

```

}
}
//rs.close();
Conn.close();
%>
total: <%=total%>$
<br/>
<a href="saveOrder.jsp">submit</a>
</p>
<do type="options" label="home"><go href="http://localhost/ExtraWapImage/extraview.jsp"
/></do>
<do type="prev" label="back"><prev /></do>
</card>
</wml>

```

### 8.2.10 saveOrder.jsp

```

<% @ page language="java" contentType="text/vnd.wap.wml;charset=UTF-8" %>
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<head>
<meta http-equiv="Cache-Control" content="max-age=0" />
<meta http-equiv="Cache-control" content="no-cache" />
</head>
<% @ page import="java.sql.*" %>
<% @ page import="java.util.*" %>
<% @ page import="com.gtom.wap.*" %>
<card id='order' title='your bill'>
<p>
<%
Connection Conn = ConnLoader.getInstance().getConnection(this);
Statement stat = Conn.createStatement();
String sql = null;
ResultSet rs = null;
sql = null;
//rs = stat.executeQuery(sql);
String userMobile = (String)session.getAttribute("phone");
Vector vec = (Vector)session.getAttribute("Bill");
int total = 0;
if(vec!=null){
for(int i=(vec.size()-1);i>=0;i--){
String text = (String)vec.get(i);
String[] item = text.split("\\;");
sql = "insert into userorder(userMobile,itemID,itemName,price)
values("+userMobile+"",""+item[3]+",""+item[1]+",""+item[2]+")";
stat.executeUpdate(sql);
out.println(item[1]+" ");

```



```

out.println("price:"+item[2]+"$ <a href='orderDel.jsp?setID="+i+"'>del</a><br/>");
//out.println(text+"<br/>");
try{
total += Integer.parseInt(item[2]);
}catch(Exception e){
}
}
}
//rs.close();
Conn.close();
%>
total: <%=total%>$
<br/>
the order saved!
</p>
<do type="options" label="home"><go href="http://localhost/ExtraWapImage/extraview.jsp"
/></do>
<do type="prev" label="back"><prev /></do>
</card>
</wml>

```

### 8.2.11 orderDel.jsp

```

<% @ page language="java" contentType="text/vnd.wap.wml;charset=UTF-8" %>
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<head>
<meta http-equiv="Cache-Control" content="max-age=0" />
<meta http-equiv="Cache-control" content="no-cache" />
</head>
<% @ page import="java.sql.*" %>
<% @ page import="java.util.*" %>
<% @ page import="com.gtom.wap.*" %>
<card id='orderDel' title='del'>
<p>
<%
try{
int setid = Integer.parseInt(request.getParameter("setID"));
Vector vec = (Vector)session.getAttribute("Bill");
for(int i=(vec.size()-1);i>=0;i--){
if(i==setid){
vec.remove(i);
}
}
}catch(Exception e){
}
response.sendRedirect("order.jsp");

```

```

%>
</p>
<do type="options" label="home"><go href="http://localhost/ExtraWapImage/extraview.jsp"
/></do>
<do type="prev" label="back"><prev /></do>
</card>
</wml>

```

### 8.2.12 Connloader.java

```

package com.gtom.wap;
import java.sql.*;
public class ConnLoader{
private static ConnLoader instance;
public Connection getConnection(Object obj){
Connection conn = null;
try{
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
}
catch (Exception E){
System.err.println("Unable to load driver.");
E.printStackTrace();
}
try{
conn = DriverManager.getConnection(
"jdbc:mysql://127.0.0.1/gtom_wap?useUnicode=true&characterEncoding=gb2312&user=root&password=");
}
catch (SQLException E){
System.out.println("SQLException: " + E.getMessage());
System.out.println("SQLState: " + E.getSQLState());
System.out.println("VendorError: " + E.getErrorCode());
}
return conn;    }
public static ConnLoader getInstance(){
if(instance == null){
instance = new ConnLoader();
}
return instance;
}
private ConnLoader(){ }
}

```