

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

Investigations on User Interface for Online Shopping: Towards a RealThing Based Design Approach

RAZIBUL HAQUE

**A MAJOR REPORT
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE**

**PRESENTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE**

**CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA**

AUGUST 2000

© RAZIBUL HAQUE, 2000



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-54333-1

Abstract

Investigations on User Interface for Online Shopping: Towards a RealThing Based Design Approach

Razibul Haque

E.ComKit is a collection of RealThing widgets that aims to make the design and prototyping of user interfaces for an online shopping mall significantly easier while supporting a new real world-based user interface model. E.ComKit widgets include shelves, product, product viewer, bag, cash, payment, delivery, etc. For making virtual shopping over the Internet, the user can manipulate and interact directly with these widgets as he does in a real-world shopping mall. Each widget dynamically and automatically varies its look-and-feel according to the cultural environment and user stereotype. Our objectives were to develop a prototype of E.ComKit based online eBookStore, and evaluate its usability compared to current online eBookStore, as well as to build specification of E.ComKit.

To my children: Zarin-Tasnim, Rifaa-Samiha, and Taqi-Tahmid Haque

Acknowledgments

I am grateful to my supervisor Dr Ahmed Seffah for his valuable advice and guidance in my study, and research work at Concordia University. Also I express my sincere thanks to all of my colleagues of Research and Development group of Dr Ahmed Seffah for working together in a friendly environment. I dedicate my love and affectionate to my wife “Mrs Showkat-Ara Minoti” and my lovely children “Zarin Tasnim, Rifaa Samiha, and Taqi Tahamid Haque” for providing me their support by maintaining high patience, while I was working at the University to achieve success of my study, and research work. I express special gratitude to my respected mother “Mrs Rezia Begum”, who had given me the valuable advice of my future life.

I render my gratitude to Dr T. D. Bui, Graduate Programme Director, Dr V.S. Alagar, Dr J. Opatrny, Dr W. Jaworski, and all of my Professors at Concordia University, who had given me an advice during study at Concordia to achieve success of my study. My respects, best wishes, and gratitude to Dr Hasan Jamil, who encouraged me to apply at Concordia University for admission to get higher education in Computer Science.

I express my gratitude and best wishes to Dr Rilling Juergen, who examines my research paper. Finally, I render my gratitude to the Chair of the Department of the Computer Science, and Dean of the Faculty of Engineering and Computer Science for their supports to study in friendly atmosphere to carry-on and pursue my studies in the Department of Computer Science.

Contents

List of Tables ix

List of Figures ix

Chapter-1

1. Introduction and Project Motivation	1
1.1 Limits of Windows, Icons, Menus, and Pointers (WIMP)	1
1.2 The Case of Web-Applications	2
1.3 RealThing-Based Design Approach for Web Application	3
1.3.1 Focus on Content, and not in Control	4
1.4 Current Web Development Tools	5
1.5 Objectives of this Research	6

Chapter-2

2. E.ComKit Concepts, Foundations and Widgets	7
2.1 Introduction	7
2.2 E.ComKit Design Anticipation	8
2.3 RealThing Widgets in E.ComKit	9
2.4 Discussion and Description of E.ComKit Widgets	11

2.5 RealWorld Interaction Style in a physical shopping mall	24
2.6 Direct manipulation of buying product(s) over an Internet	25
2.7 Context of Use Pluggable Look-and-Feel	28

Chapter-3

3. An online eBookStore Prototype	31
3.1 Introduction	31
3.2 Design layout of an online eBookStore	31
3.3 E.ComKit based user interface of an online eBookStore	33
3.4 Traditional GUI based User Interface of an online eBookStore	35
3.5 Evaluations of Traditional GUI based, and E.ComKit based eBookStores according to Norman Principles	36
3.6 Feedback and comments after evaluating both eBookStore interfaces	40

Chapter-4

4. E.ComKit based eBookStore OO Modeling	42
4.1 Introduction	42
4.2 Component-based E.ComKit RealThings User Interface	43
4.3 The E.ComKit Modeling for an online eBookStore	43

4.4	E.ComKit Object-Orientation	44
4.4.1	Assumptions	44
4.4.2	Generalization and Specialization	45
4.4.3	Product to Cash	46
4.4.4	E.ComKit based an eBookStore Class Details	47
4.5	E.ComKit based an eBookStore library and relationship between classes	50
1.	Promotion shelf [Product(s) Shelf]	50
2.	User Shelf [Product(s) Shelf]	50
3.	Shelf of shelves (Main Shelves)	51
4.	Shelf of Product [Sub shelves, Products shelf]	51
5.	Product to display details information in a window.	51
6.	Interactive ToolBars (Interactive container)	52
6.1	Bag:	52
6.2	Cash:	52
6.3	Invoice:	53
6.4	Payment:	53
6.5	Bill:	53
6.6	Delivery:	53

6.7 Advisor:	54
6.8 Shopping Record:	54
6.9 User Data:	54
4.6 An eBookStore Object Oriented Class Diagram	54

Chapter-5

5. Lessons Learned and Perspectives	56
5.1 Conclusion and future work	56
5.1.1 Customization of an online shopping environments	57
5.1.2 Changing Look and Feel of RealThing Interface Object	59
5.2 Discussions and Achievements	61

List of Tables:

1. E.ComKit based an online eBookStore widget's look and feel	10
2. Context of Use pluggable Look-and-Feel and its behavior	29
3. Evaluation of Amazon.com, and E.ComKit based eBookStore	40

List of Figures:

1. Shopping interaction styles in a physical shopping mall	24
2. "Drag and drop actions" for buying products	27
3. Context of use pluggable look-and-feel examples	30
4. Design layout of an eBookStore shopping mall	32
5. E.ComKit based an online eBookStore	34
6. Traditional GUI based an online eBookStore such as Amazon.com	35
7. Shelf is a generalization	45
8. Product to cash	46
9. An eBookStore OO class diagram	55
10. Changing look and feel of RealThing objects	60
11. E.ComKit based an eBookStore model	62
12. List of Objects of eBookStore model	63

Appendix-A	62
1. E.ComKit based an eBookStore Model	62
2. Each Widget is an object of an eBookStore Model	63
Appendix-B	64
1. Java source codes (as sample examples) of an online eBookStore	64
2. HTML, DHTML and Java scripting languages (as sample examples) to develop an online eBookStore prototype interface	102
Reference	113

Chapter-1

1. Introduction and Project Motivation

1.1 Limits of Windows, Icons, Menus, and Pointers (WIMP)

The user interfaces of today are dominated by the so-called WIMP UI Model - Windows, Icons, Menus, and Pointers. While there is no denying the success of these interfaces in bringing desktop computing to millions of users across the world, the GUI has grown to be a cluttered, discordant world of clashing icons and wasted screen space.

In the WIMP world, objects (or more usually, widgets) are presented in rectangular windows. They do not look real, or even bear more than an occasional passing resemblance to anything in our real world outside the computer. And amongst the visual noise and clutter, are hidden the clues necessary to make the cognitive leap to accommodate a metaphor which relies on the idea that 'windows' can exist on a 'desktop.' Objects and applications alike are represented by icons. But these icons only show a gross level of information - they indicate the class of object, but rarely impart status information or make important properties apparent. All icons are the same relative size, and pretty much the same shape.

In the WIMP world, menu bars and tool bars proliferate. Users spend much of their time 'mining' menus or hovering over buttons waiting for help. As the user fumbles through the interface in search of particular functions, the clutter and visual pollution detracts from the content or task that should be

the user's main concern. In such cases, the user interface itself distracts and intimidates the user rather than helping.

1.2 The Case of Web-Applications

If 'suite-bloat' is one driver towards a clearer, more accessible world for the users, the World Wide Web is another. The Web is content-based, and that content assumes only the relatively limited built-in function of a Browser. Any other functionality has to be provided intrinsic to the content. Often this function takes the form of pseudo-menus and pick-lists, but even this exhibits a freedom and a freshness of design. This is partly because of the fact that the content is freed from the software and mental constraints of the standard UI tools and widgets provided by the underlying platform.

It is also partly due to the democratization of content. The pages of content on the Web are often put together by designers and novices with little or no previous computer design or programming experience. But, in fact, these are people who are psychologically more able to communicate with the 'common user'. Their designs are not polluted by the design constraints that the UI designer used to have to deal with.

The most interesting design emerging from the Web, though, is where function is intrinsic to content, - where objects (for want of a better term) exhibit exactly the behavior that their visual appearance suggests; where function is in support of task; where form suggests purpose; where interaction is apparent from visual appearance.

By transferring Web to WIMPs, It is hardly surprising that there is a general move to incorporate web-like interfaces in the WIMP world. But generally, this is a patchy solution that combines two fundamentally incompatible styles of UI.

1.3 RealThing-Based Design Approach for Web Application

In many respects, the way to forward for today's content-centric world is obvious. Cut down on the visual clutter and pollution and focus instead on the content. This will have the bonus effect of making that content seem more like the 'real world' content as traditional design methodologies and experience can be brought to bear. An escape from the constraints of the rectangular world of the WIMP interface can only bring the UI closer to the world that the user already knows and understands.

In fact, we can already see the start of a reaction against the cluttered WIMP world. Lotus SmartSuite, for example, provided tool bars according to context, rather than present every conceivable function for every possible circumstance.

More interesting is the UI demonstrated by Lotus Organizer, where most operations can be performed in place without resorting to menus or tool bars. Users are presented with and or/ by UI that resembles something these are already be understandable - a book. The Smart Center drawers for Calendar and address are also a natural and very usable extension of this philosophy. They are unobtrusive yet accessible - combining the value of pull-down menus with real objects that provides valuable, but not over-comprehensive, function for standard and frequent tasks to the user.

1.3.1 Focus on Content, and not in Control

While the general movement is in the right direction, the progression is still quite slow. We can take these notions further still, and the key to, this is IBM RealThings [8] based on easy-of-use. RealThings exhibit a new, real-world user interface style. They set a new direction in making user interfaces more approachable for novice and casual users. At its simplest, the RealThings philosophy is to make software constructs and applications appear as they would if they existed in the real world outside the computer. Object mechanisms are presented in context, and actions are surfaced in more natural ways.

The key for making RealThings works as the next major stage in the development of ease of use system, and to have a knowledge transfer from the real world to virtual world. In such cases the computer user feels more natural and can adapt the system easily by interact a RealThing based interface object.

By making the behavior of virtual objects more natural and intuitive - more like the user expects from his or her experience from the RealWorld object by doing a physical interaction - we make it more accessible, less daunting, easier to comprehend and to use. In short, rather than expecting the user to learn what is in effect a new language, we leverage what the user already knows. And in doing so, we shift the focus away from controls and back on to content.

1.4 Current Web Development Tools

Presently, the available current tools for web application such as CyberStudio, FrontPage, and PageMill etc are using to design HTML based web pages that vary the degrees of success, and most HTML editors offer tools that enable you to create tables. The drawback is that their interfaces will slow you down, making your editing sessions less productive than they could be [16]. And the current web tool has limitations to match the real-world object, while the user makes interaction to the interface object over the Internet as he or/ she does shopping in the RealWorld shopping mall.

The facts of web applications including eCommerce are content-oriented. And we find that the traditional GUI widgets using for online shopping are inappropriate because it has several GUIs limitations. For example, it is necessary to incorporate a special button in shopping carts (or/ bag) to remove a product. In such cases the user cannot be assumed to understand that they can ignored (or/ cancel) a purchase by buying a quantity of zero product (although this technique can also be supported, since it does not interfere with users unless they decide to use it on their own initiative) [2]. Mostly, the current web or/ eCommerce applications are being developed by using traditional GUI widgets like menus, scrollbars, buttons and text input field etc. And it is very difficult to follow the user attitude, behavior over the Internet using traditional current tool, unless the user interface object could match the RealThing object in a RealWorld shopping mall.

The design difficulties also vary based on different reasons:

- Narrowly focused on small devices for using Internet based any application, and cannot be supported full range of features because of lack of screen space or/ low bandwidth.
- The use of different guidelines for a web application with multiple user interfaces are not acceptable.

1.5 Objectives of this Research

The aim of this research is to make the RealThing based design for an online shopping significantly easier, while it supports the following innovative user interface paradigms:

- Similar to a real-world shopping approach, virtual shopping takes in place of culture that defines expectation desires, policies, values, preferences, etc.
- The virtual shopping should be developed in such a way that it can be interacted easily by the user, and automatically adapted to different cultures and user stereotypes.
- The virtual shopping should be matched with a RealWorld shopping that the user does in a real shopping mall, in such cases the user will feel more natural, and will satisfy by doing shopping over the internet.
- The user can interact with a number of high-level built-in functions to the Web or/ eCommerce application, including display products, and drag & drop mechanism, and controlling the user interface of cultural pluggable look and feel etc.

Chapter-2

2. E.ComKit Concepts, Foundations and Widgets

2.1 Introduction

The E.ComKit (Table-1) is a set of RealThings widgets that exhibit a new real-world user interface style. The RealThings widgets aims to make the user interfaces more approachable for novice and casual users. The key to make RealThings work is transferring knowledge from the real world to virtual world. And this approach introduced by IBM research as a solution of GUIs limitations.

In the E.ComKit approach, the virtual shopping takes place in a context that defines expectations, desires, policies, values, preferences etc. Therefore, a virtual shopping environment and especially its interface should be developed in such a way that it can be automatically and easily adapted, by the system or/ by user, to the different context of use as per technical, cultural and social dimensions etc. It is important that the eCommerce or/ web application will be so successful if everyone can customize interface of virtual shopping mall easily to make a RealThing based virtual shopping over the Internet. Therefore, the aim of E.ComKit is to make interaction easily to web interfaces for customization of virtual shopping mall, and or/ by creating his /her own user interface, or/ redesign or/ reorganize interfaces, and changing look and feel based on cultural environment, and user stereotype. The E.ComKit approach defines to make it possible for users to customize interface easily over an Internet browser.

2.2 E.ComKit Design Anticipation

The E.ComKit aims to:

- Develop rapidly different design solutions,
- Help non-expert designer to design and implement RealThing-based interfaces,
- Facilitate the portability of user interface in different environment,
- Help to evaluate the user interface,
- Improve easy-to-use interfaces, easy-to-learning, easy-to-adapt,
- Bring facilitate customization of user interface by redesigning, or/ reorganizing, or/ creating a new one, or/ changing pluggable look-and-feel according to the context of use and user stereotype (Table-2 and Figure-3)

2.3 RealThing Widgets in E.ComKit

At its simplest, the RealThings philosophy is to make a virtual shopping mall appear as it currently exists in the real world outside the computer.

Products are presented in context, and interactions are surfaced in more natural ways. By making the behavior of user interface widgets more natural — more like what users expect from their experience of the world — we render it more accessible, less daunting and easier to comprehend and use.

However, RealThings are more than simply copying the real world concept with its inherent limitations; and it provides navigation techniques in natural way that enhance the real world, as well as offer mechanisms and interaction cues that are easily recognizable and understandable.

The user interface is a collection of widget(s). At its current moment, the E.ComKit propose a set of RealThing widgets in *Table-1*, which are used for an online eBookStore prototype development as follows.

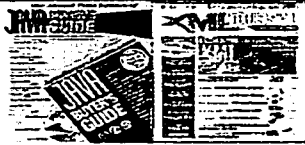




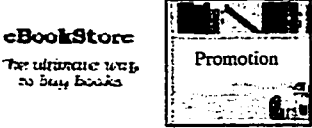





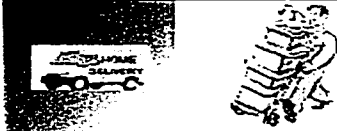



RealThings Object	<i>Look-and-Feel from an Online eBookstore</i>	RealThings Object	<i>Look-and-Feel from an Online eBookstore</i>
1. <i>Product</i>		2. <i>Product Viewer</i>	
3. <i>Shelf of Shelves</i>		4. <i>Shelf of Product(s)</i>	
5. <i>User Shelf</i>		6. <i>Promotion Shelf</i>	
7. <i>Bag</i>		8. <i>Cash</i>	
9. <i>Invoice</i>		10. <i>Payment</i>	
11. <i>Bill</i>		12. <i>Delivery</i>	
13. <i>Advisor</i>		14. <i>Shopping Record</i>	
15. <i>User Data</i>			

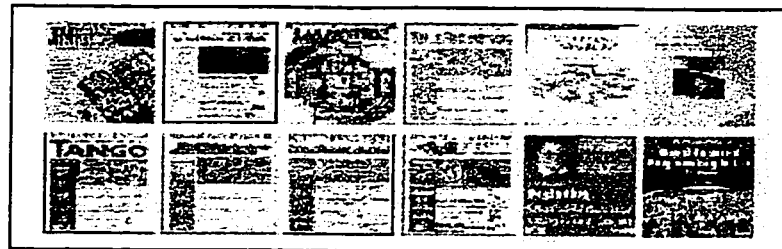
Table 1: E.ComKit based an online eBookStore widget's look and feel

2.4 Discussion and Description of E.ComKit Widgets

A RealThing widget has two states, generally a close state, which means that the widget is not active (i.e. the interaction has not been done on it yet by the user). The open state means that the widget is active (i.e. the interaction has been done by the user). That means, when the user interacts on any widget with a mouse pointer, its change the state i.e. from closing state to an open state and vice versa.

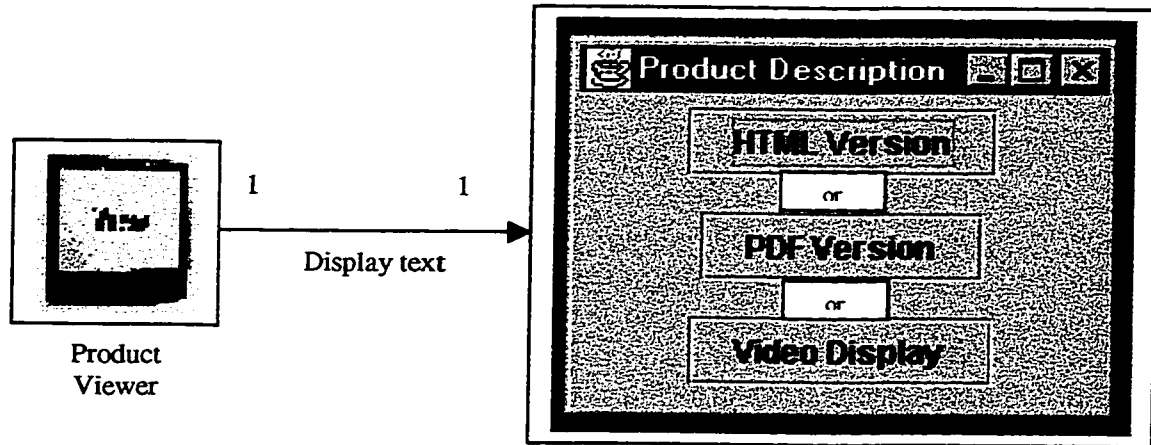
The following are the list of E.ComKit widgets for an online eBookStore.

1. A *Product* is anything that the users need to purchase. For an online eBookStore, it can be a book, magazine, newsletter etc. The user moves the mouse pointer over the product or/ drag and drop product to a product viewer for displaying product's details.

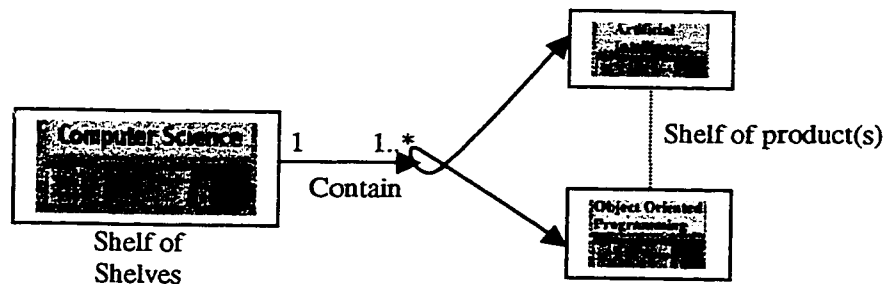


List of Products

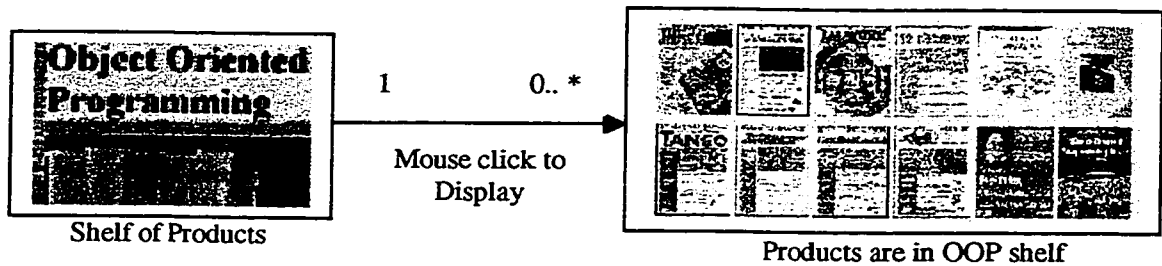
2. A *Product Viewer* displays product information (i.e. Product Title, author, description of product, and price, etc.) in a window by using different formats and medium, such as HTML, PDF, Video, etc. The user needs to drag and drop the product to a product viewer to display detailed product's information in different format.



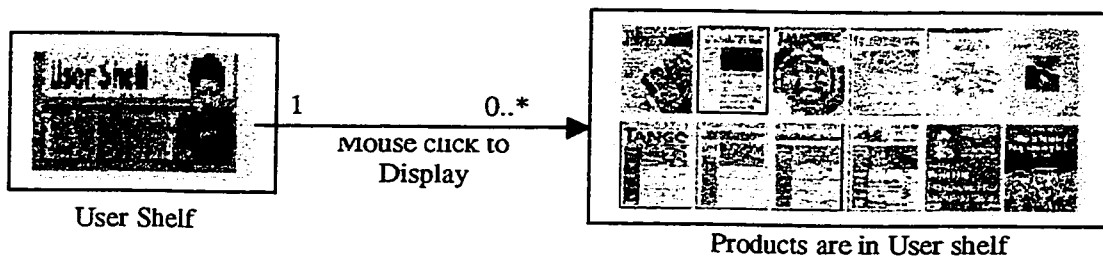
3. A *Shelf of Shelves* means in turn it contains different sub shelves, and with each one (sub-shelf) is representing a specific domain area, such as Computer Science Shelf, which contains Object Oriented Programming shelf, Artificial Intelligence shelf etc.



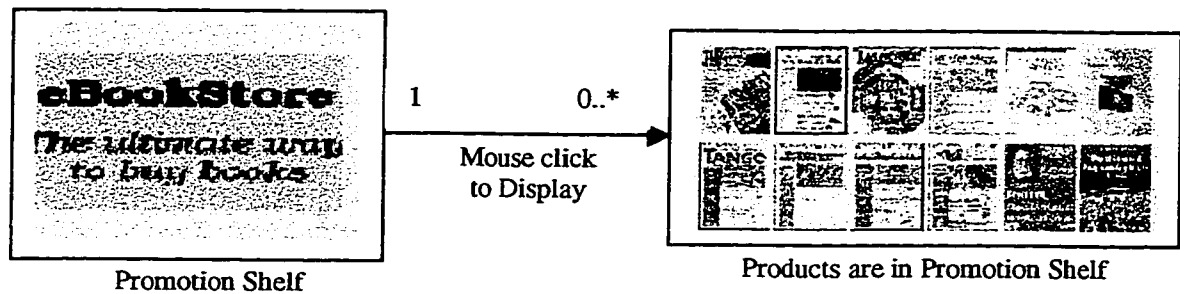
4. A *Shelf of Products* organized by subject is a collection of products. The shelf is the only way to organize a group of products. Each shelf is related to a specific subject, such as Object Oriented Programming shelf contains all books related to its subject area i.e. Java, C++ Programming etc.



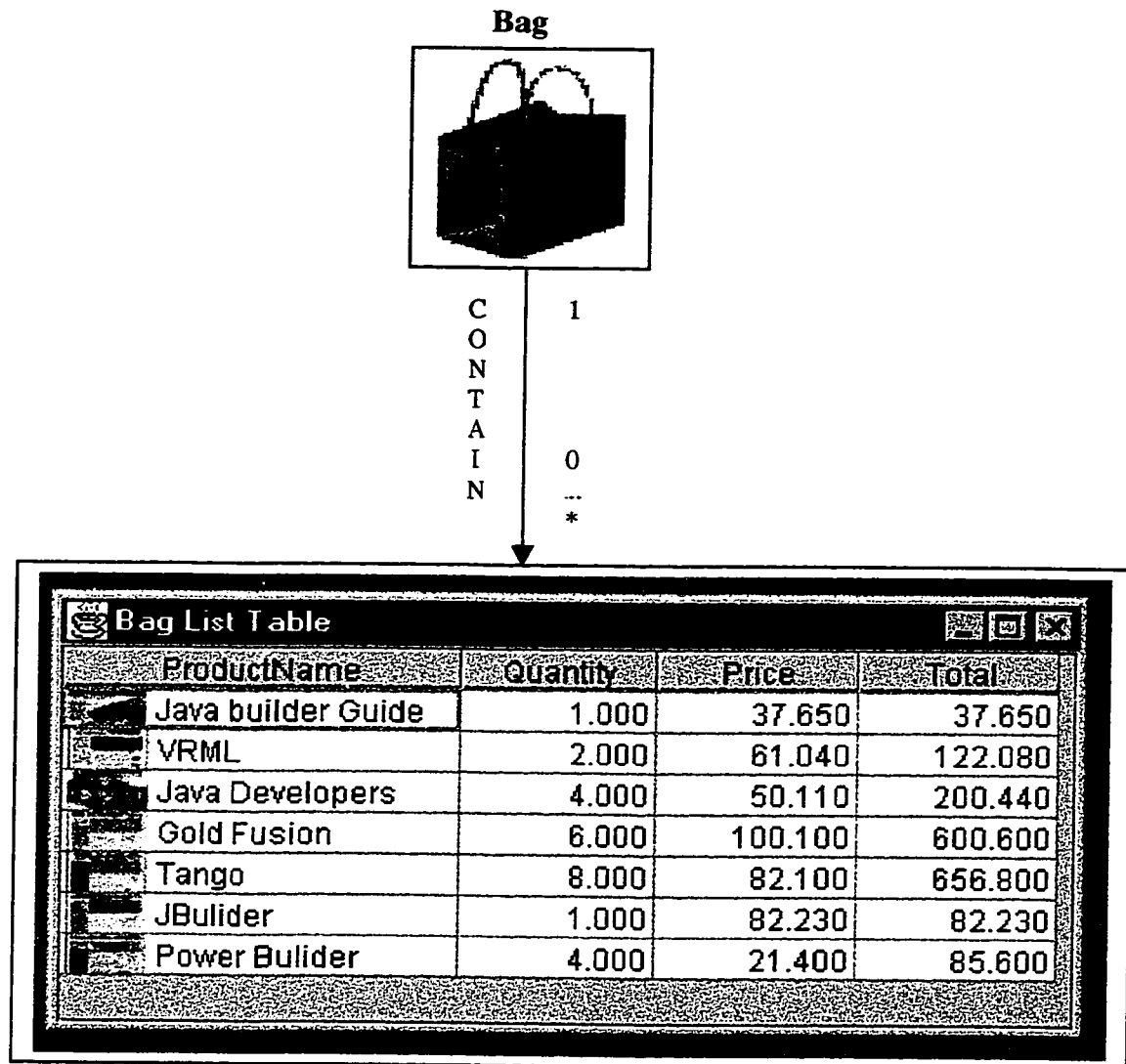
5. A *User Shelf* is a specialized shelf. It contains a collection of products that is created, or/ selected on the fly by the users. A User Shelf is an example, that the user can customize an online shopping mall by creating or/ changing its look and feel.



6. A *Promotion Shelf* is a shelf that contains a collection of products on sale and/or related to the user's preferences. When Iconized, the promotion shelf looks like a banner of an online shopping mall. When open, it looks like any other shelf that contains product(s). The eBookStore manager decides which products are on sale according to preferences entered by the user or automatically captured from shopping record by the system.

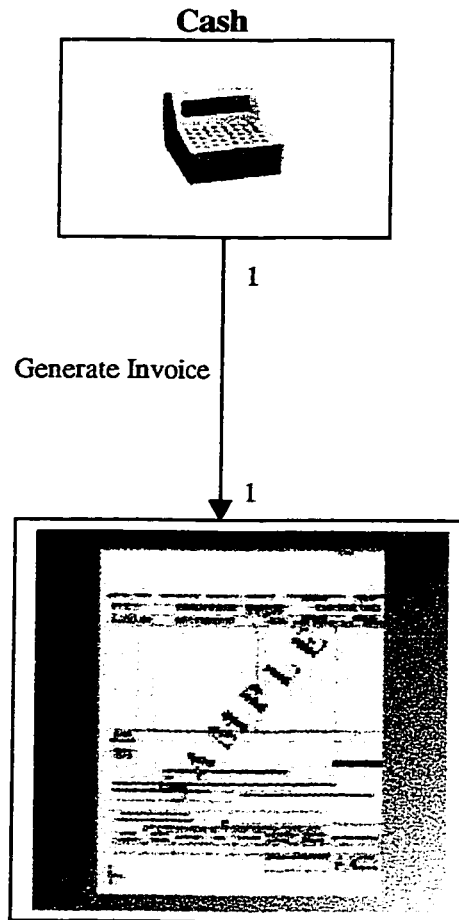


7. A *Bag* is an unordered temporary shelf that contains list of products that the user has selected for buying. The user can add or/ remove product(s) to or/ from bag using drag and drop mechanism. The method is applied to the bag that adds or/ remove the specific product to or/ from a bag using drag and drops mechanism, and displays the products that are now available in a bag instantly.

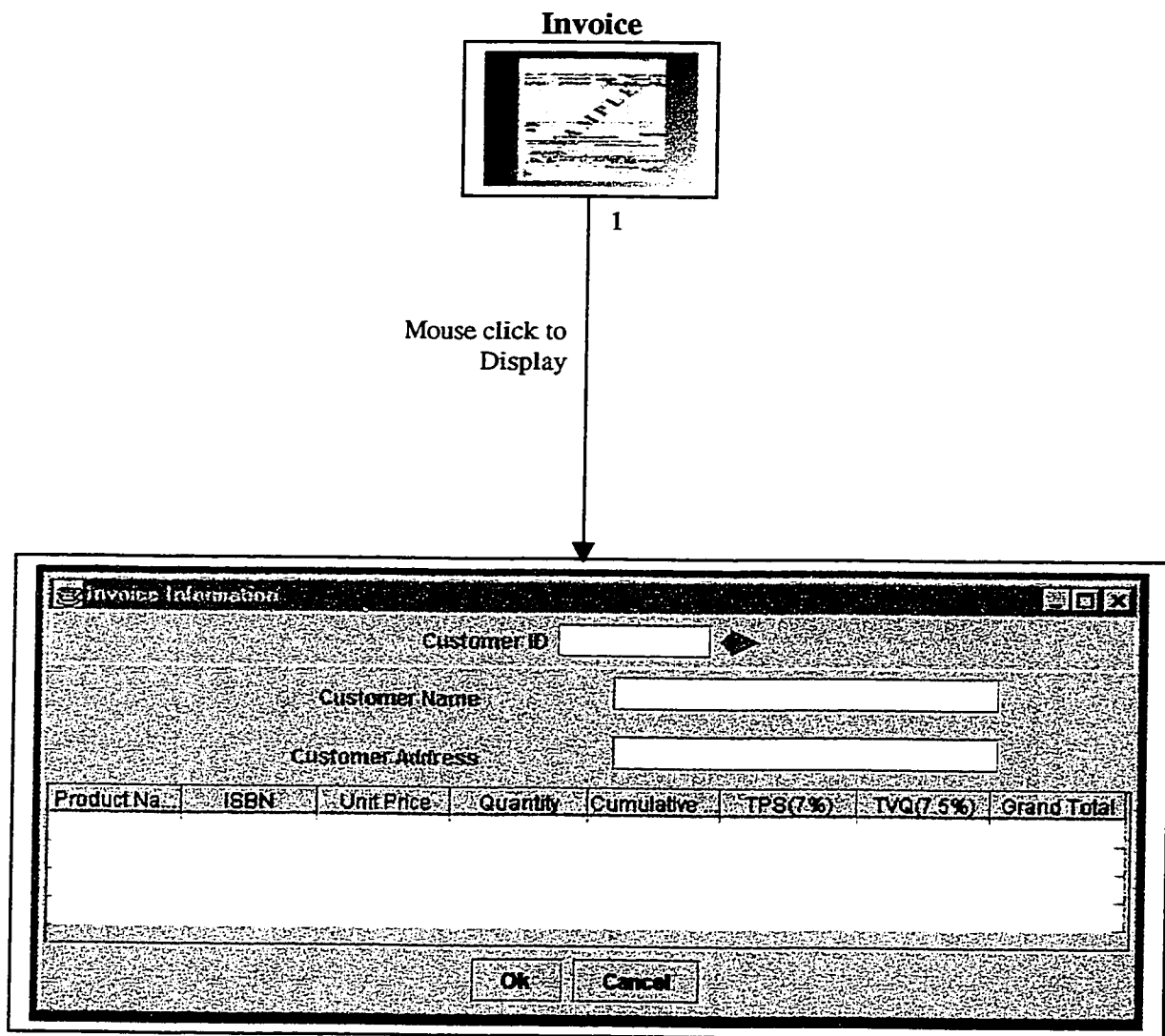


Products are in a Bag

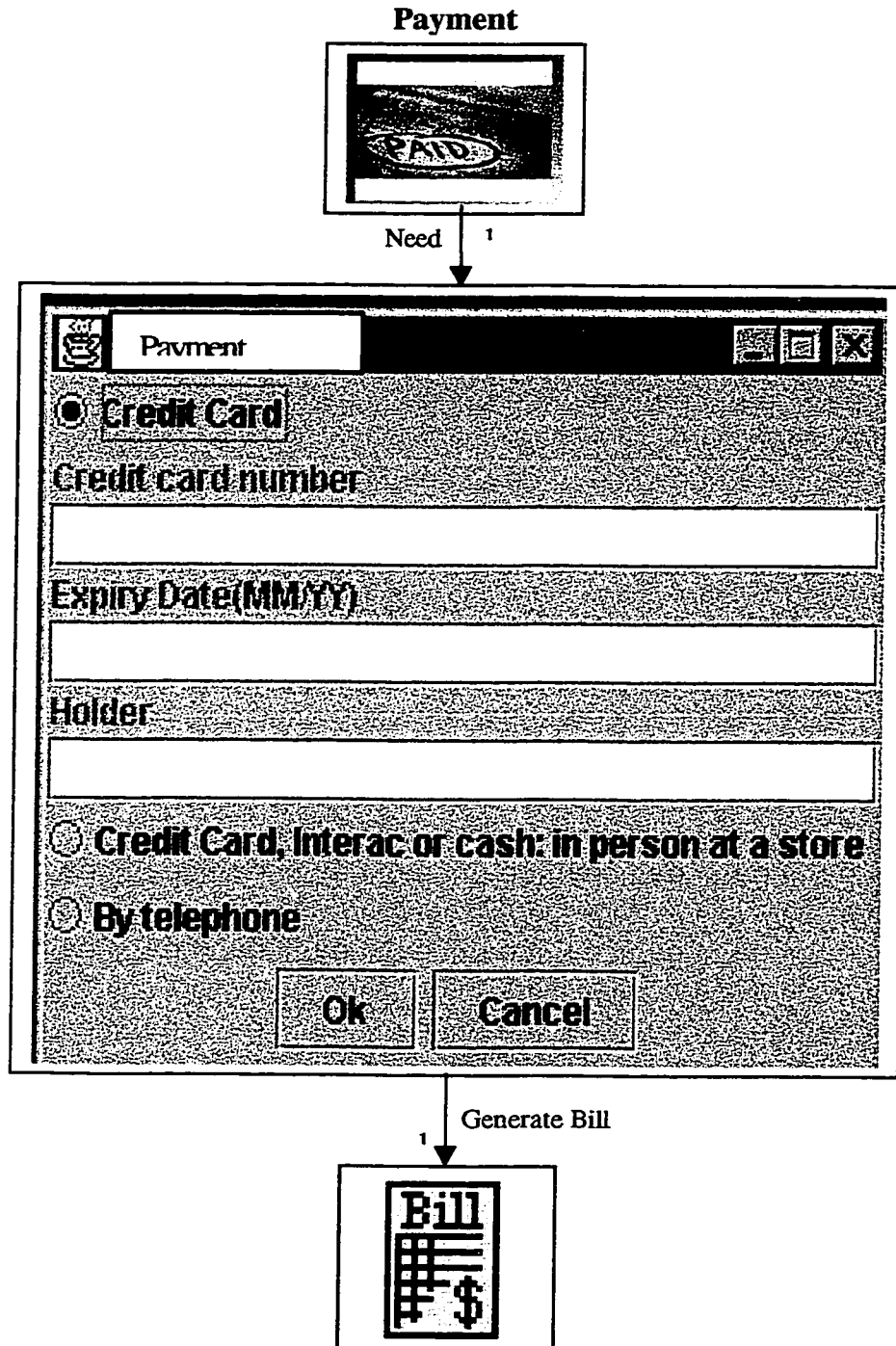
8. A *Cash* is a register machine that counts and cumulates the product's price. When the bag is dragged and dropped into a *Cash*, its counts and cumulates prices, and generates an invoice automatically.



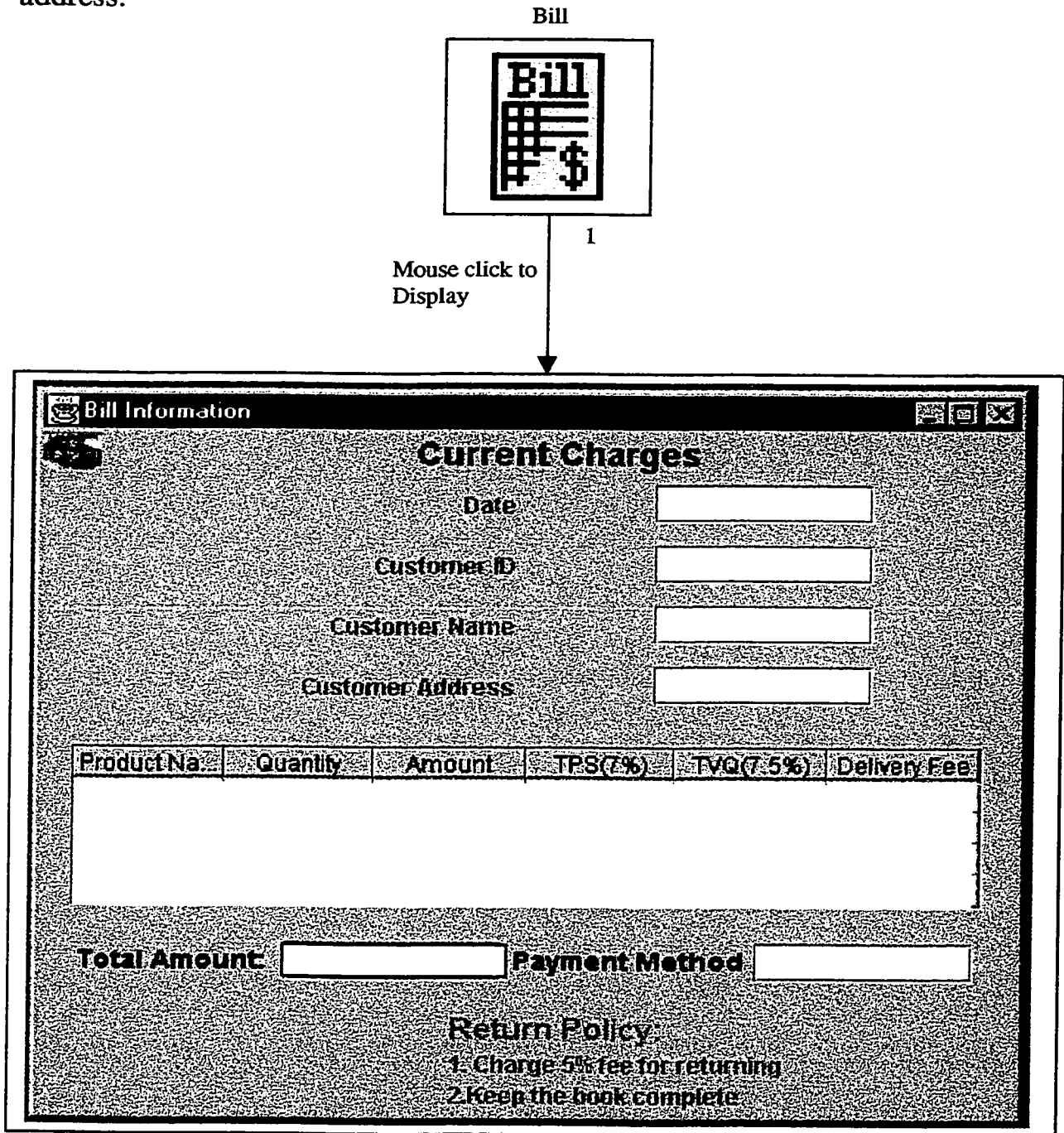
9. An *Invoice* is a list of products that the user already selected for buying over an internet and it includes product title, ISBN, price, cumulative total, applicable taxes, grand total price, customer name, address etc. Drag and drop an Invoice into a payment widget to pay the product price.



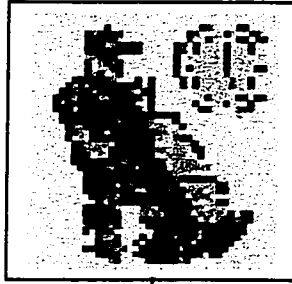
10. An online *Payment* is required to have the detailed information of card (or/ eCash or/ Interact) to buy product(s) over an Internet. For example credit card type, card no, expired date, name of cardholder etc. And as soon as the payment is done, it generates a bill automatically for product(s) delivery.



11. A *Bill* is a receipt of payment that includes customer ID, name, address, list of products, method of payment, etc. and product return policy. By doing drag and drop of a bill widget into a delivery widget, the user can complete a job for delivering a product or/ a list of products to the customer specific address.



12. A *Delivery* uses a bill with the user's delivery address and personal data to arrange a delivery scheduling.

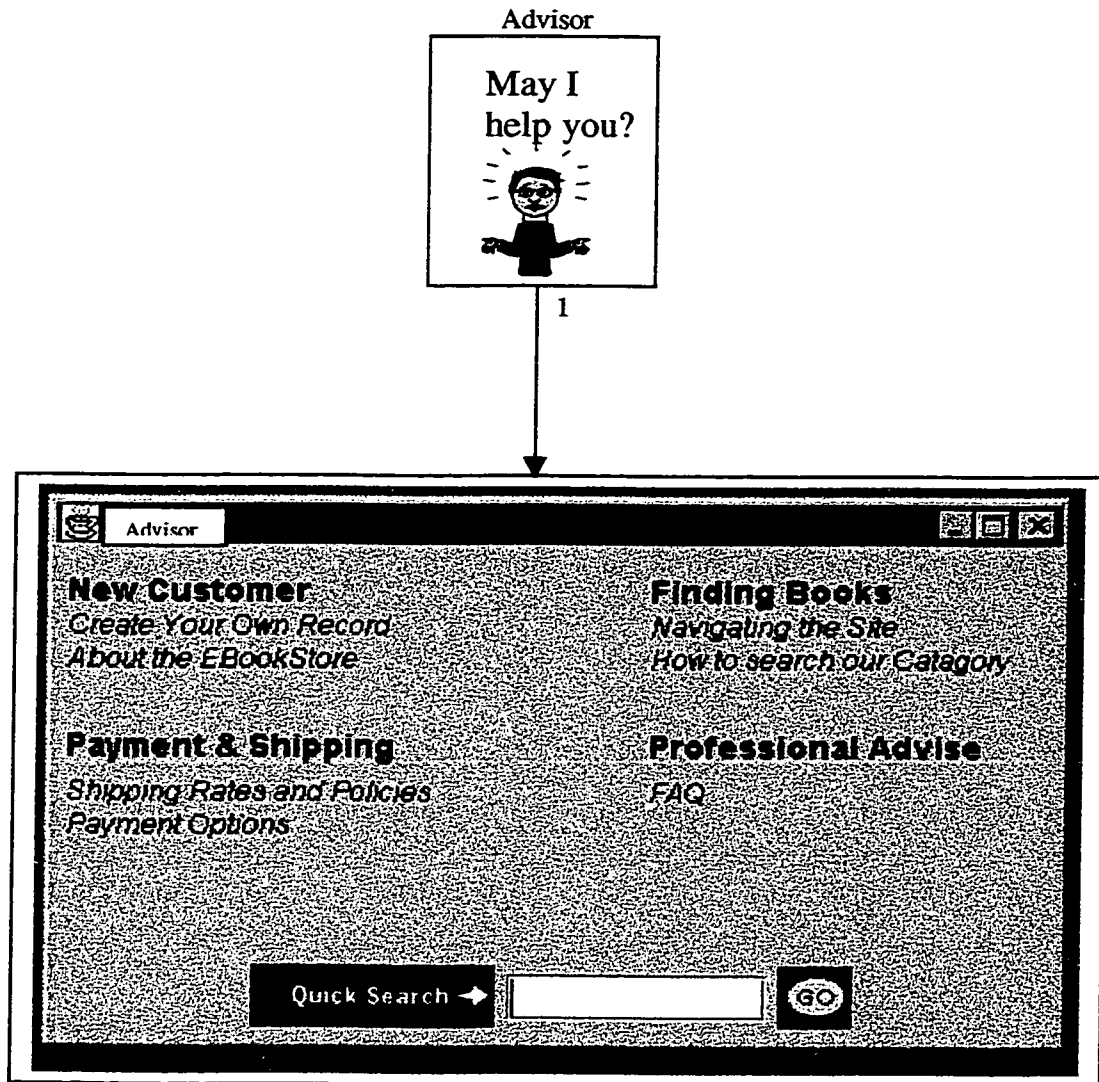


Make an arrangement

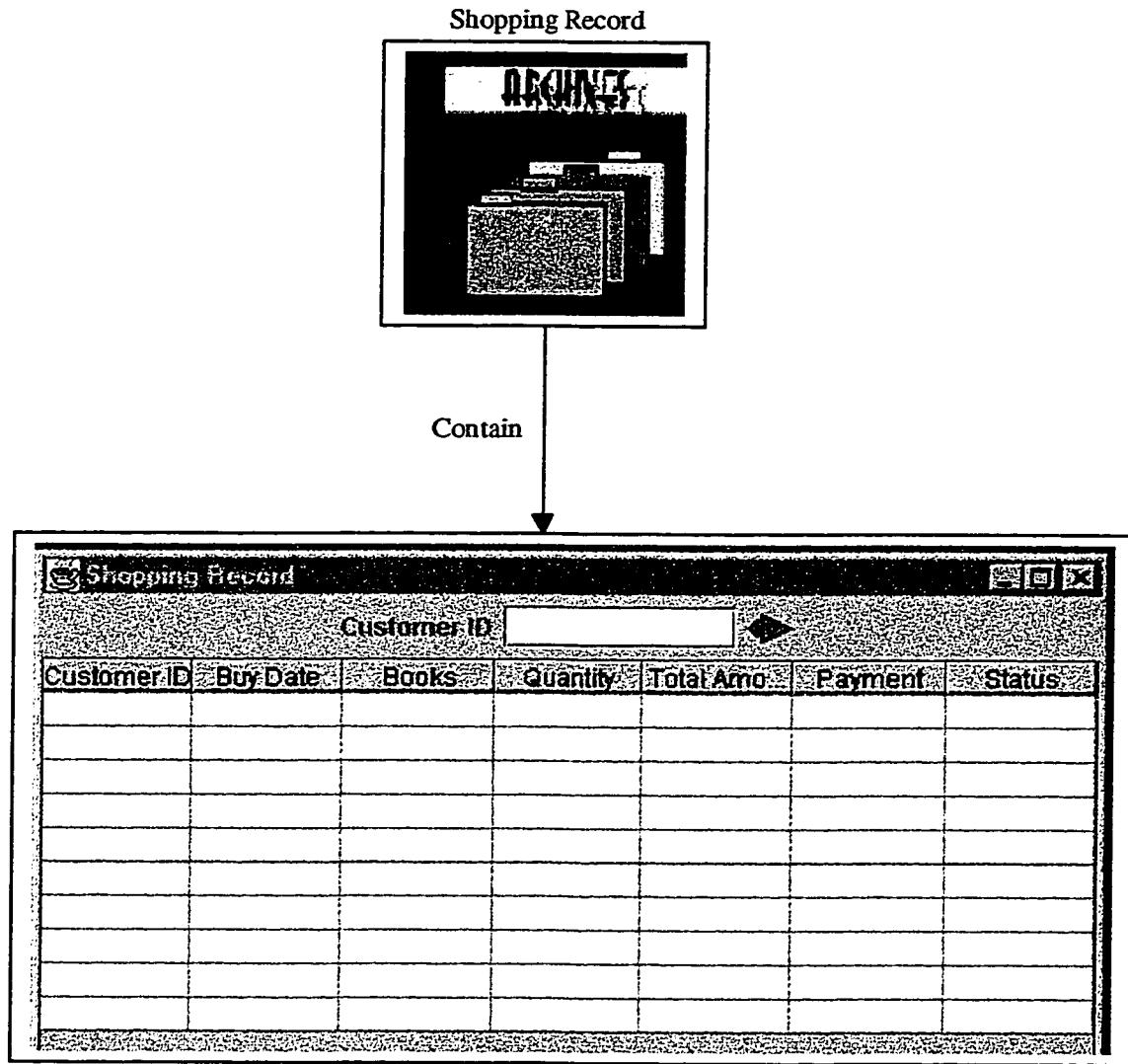
The image shows a dialog box titled "Delivery" with a standard Windows-style title bar. The dialog contains the following fields and controls:

- Customer ID:** A text input field with a right-pointing arrow button.
- Customer Name:** A text input field.
- Telephone:** A text input field.
- Alternate Telephone:** A text input field.
- Customer Address:** A text input field.
- Delivery Date:** A date picker showing "January" in a dropdown, followed by two empty input boxes for day and month.
- Delivery Time:** A time picker showing "0" in a dropdown, followed by an empty input box for minutes, and "AM" in a dropdown.
- Delivery Method:** A dropdown menu currently showing "Express Post - By fast mail".
- Different Address:** A text input field.
- Buttons:** "Ok" and "Cancel" buttons at the bottom.

13. The *Advisor* is an online agent, who can assist users to find and select product(s) from an online shopping mall for buying easily and or/ provides other online help, if the user needs it over the Internet.



14. The *Shopping Record* contains a list of invoices, bills and the user preferences of buying product(s) over an Internet. It helps the user to make new shopping easier.



15. The *User Data* (for an example of Razibul image) includes customer information such as name, postal address, telephone numbers etc. User Data is used to arrange for delivery and or/ generate an invoice and or/ bill.



Contain user's personal information

A screenshot of a software dialog box titled "User Information". The dialog box has a standard Windows-style title bar with a logo on the left and minimize, maximize, and close buttons on the right. The main area contains a list of text labels on the left and corresponding empty text input fields on the right. The labels are: "First Name", "Middle Name", "Last Name", "Street", "City/State/Province", "Country", "Post Code", "Area Code", "Telephone Number", "Password", and "Reenter Password". At the bottom of the dialog box, there are two buttons: "Ok" and "Cancel".

First Name	<input type="text"/>
Middle Name	<input type="text"/>
Last Name	<input type="text"/>
Street	<input type="text"/>
City/State/Province	<input type="text"/>
Country	<input type="text"/>
Post Code	<input type="text"/>
Area Code	<input type="text"/>
Telephone Number	<input type="text"/>
Password	<input type="text"/>
Reenter Password	<input type="text"/>

Ok Cancel

2.5 RealWorld Interaction Style in a physical shopping mall

In real shopping mall, the people are tempted to pick-up product(s) from a shelf, and put-into a bag or/ cart for buying, i.e. the customer chooses product(s) and pick-up product(s) from a shelf and put-into a bag, then bring bag into a cash for buying (Figure-1).

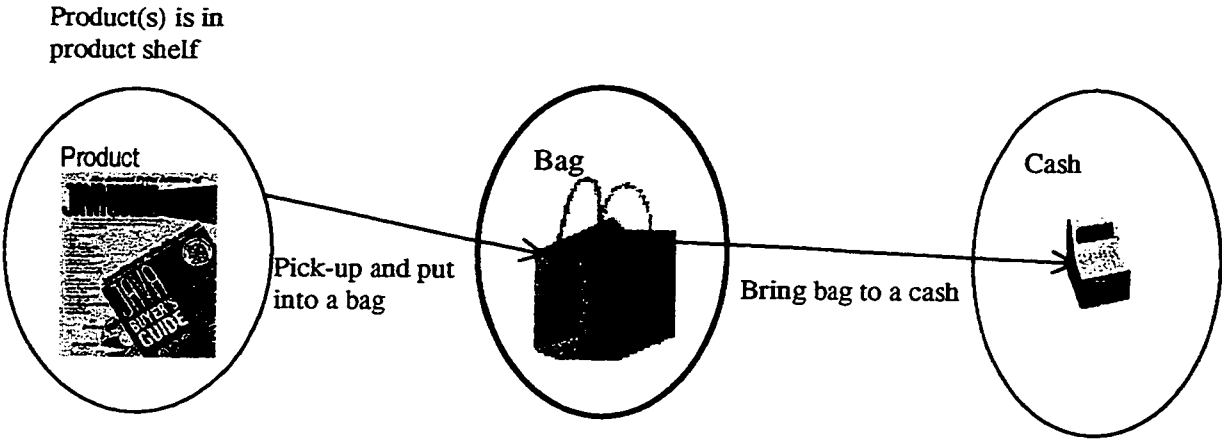


Figure-1: Shopping interaction styles in a physical shopping mall

2.6 Direct manipulation of buying product(s) over an Internet

For an online virtual shopping mall, products are displayed in a window by an open product shelf. The user can drag and drop product into a bag, And drag and drop bag into a cash and so on.

For example - to buy product(s) from an online eBookStore, the user can follow the dragging and dropping mechanism by using mouse pointer:

- To buy product(s) from different shelves, including promotion shelf and user shelf, the user used mouse click to the product shelf for displaying product(s) image in a window.
- To know the detailed information about the product the user needs to move the mouse pointer over the product image, or/ drag product image and drop into a product viewer (Figure-2)
- Choose or/ select product by using mouse pointer, and drag and drop product(s) into a bag (Figure-2)
- Product(s) that is already in the bag, then dragging and dropping bag into a cash to generate an invoice automatically (Figure-2), and mouse click on an invoice to display details invoice information.
- Dragging and dropping an invoice into a payment to generate a bill automatically (Figure-2), and mouse click on a bill to display details bill information.

- While the payment is done, the bill is generated automatically, and dragging and dropping a bill into a delivery widget to make the order completed, and prepares delivery time scheduling, and print user's delivery address to the delivery order sheet for delivery on time, Figure-2
- To get details about specific product(s) of any time that have already been in purchased, (or/ user preferences), used mouse click on shopping record widget to get an user's shopping record (Figure-2)

So, the important feature of E.ComKit approach is that the user can manipulate and interact directly with the RealThing widgets by using direct dragging and dropping mechanism to buy product(s) over the Internet.

Figure 2 Illustrates the various kinds of interactions that the user can follow to complete actions for buying product(s) over the Internet by using dragging and dropping RealThings widgets with the mouse pointer, [like the user used to habituate in Real World shopping mall to buy a product(s)].

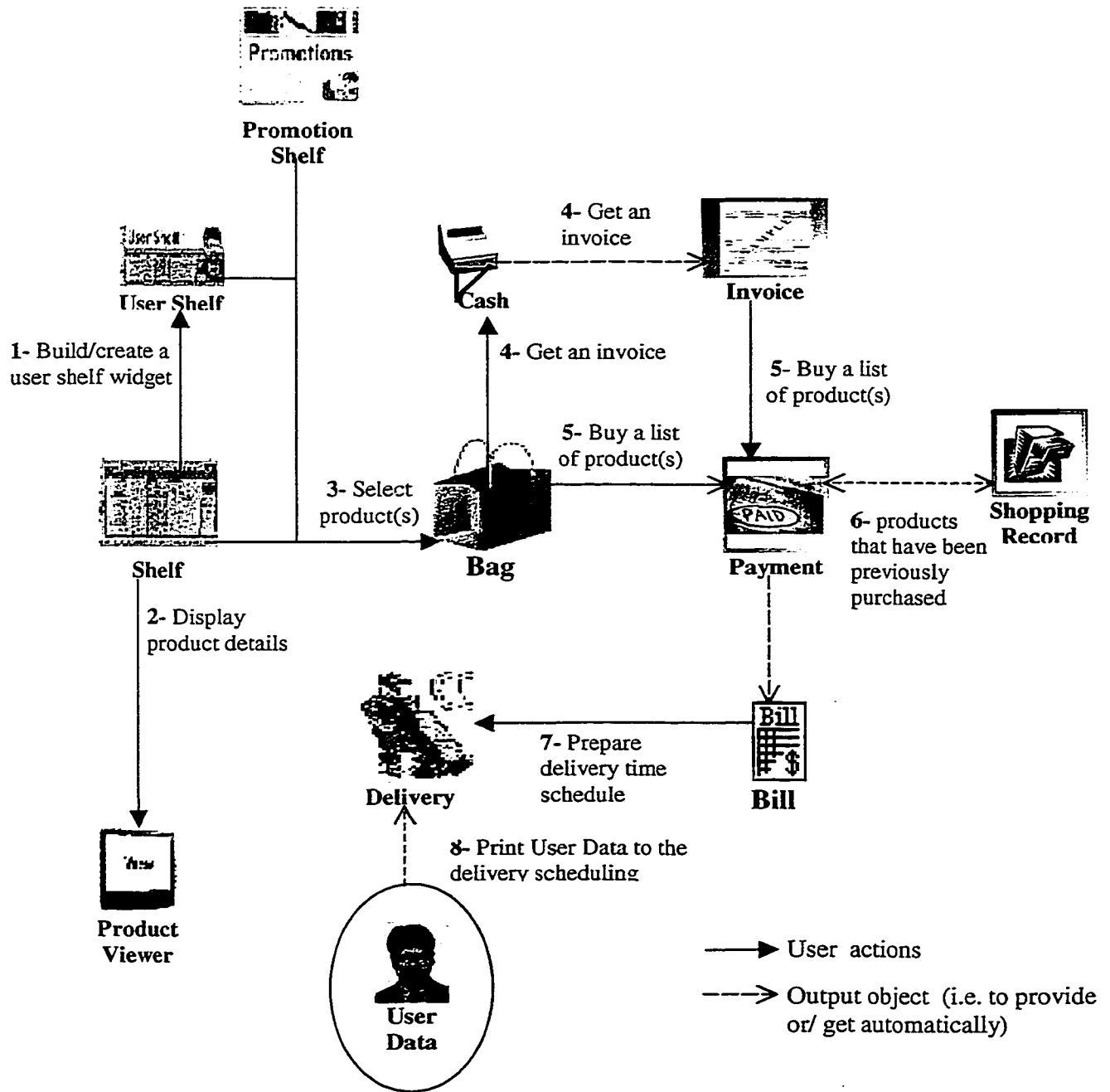


Figure 2: "Drag and Drop Actions" for buying products

2.7 Context of Use Pluggable Look-and-Feel

The context of use pluggable look-and-feel is an extension and generalization of the pluggable look-and-feel introduced by Java Swing [Walrath-99]. This mechanism allows an interactive application to choose the appearance and behavior (i.e. look-and-feel) of UI components according to the technical environment including hardware and operating system and its social and cultural dimension, where application will run.

Context of use pluggable look-and-feel provides essential information to user, who is involved in creating or/-redesigning or/-reorganizing user interface. In particular, the E.ComKit research work offers to user how to redesign and changing the pluggable look and feel of user interface widget, as per context of use and user stereotype over an Internet.

With the context of use pluggable look-and-feel, the user interface varies according to user characteristics, attitude, tasks based on social and cultural dimension as well as technical environment (Table-2).

By clicking any of the interface objects, it display a set of RealThing widgets that the user can select one as per context of use and user stereotype to change the existing one. Here are the main components of context of use in general as follows (Table-2):

Components	Attributes	Example of values
<i>Behavioral</i>	<i>Preferences</i> <i>Hobby</i> <i>Gender</i> <i>Disable</i> <i>NotDisable</i> <i>Age</i>	<i>Buying books</i> <i>Reading Nobel</i> <i>Male/Female</i> <i>Need WheelChair</i> <i>Car and or/ Can Walk</i> <i>Childern, Adult, Oldage</i>

<i>Technical environment</i>	<i>Operating system Hardware</i>	<i>Windows/NT Unix, Linux PC Pentium</i>
<i>Social dimension</i>	<i>Environment</i>	<i>Home Office School Market Community center Industry Village Town or/ city</i>
<i>Cultural dimension</i>	<i>Language Semaphore (symbol) Color Religion</i>	<i>English French Arab Bangla Chinese Persian Continent Country Green, Red etc. Islam, Christianity, Judaism etc</i>

Table 2: Context of Use pluggable Look-and-Feel and its behavior

To change pluggable look-and-feel as per the context of use, the E.ComKit maintain the following principles as follows:

- Even if the appearance of the E.ComKit widgets changes, it must retain the same functionality.
- Each widget handles its own individual view-and-controller responsibilities.

- Each widget delegates the look-and-feel-specific aspects of its responsibilities to whatever context of use object the currently installed to provide new look-and-feel appearance.
- Users may also add or/ remove the context of use attributes, and values by executing a component based on agent program to customize his or/ her interface for displaying a new look and feel of widget appearance.

An E.ComKit-based user interface exploits one simple and well-defined conceptual model. For example, a cart or/ a basket (**Figure 3**) can represent a bag. And the look-and-feel for payment can be credit card, e-cheques or e-cash (Interact payment).

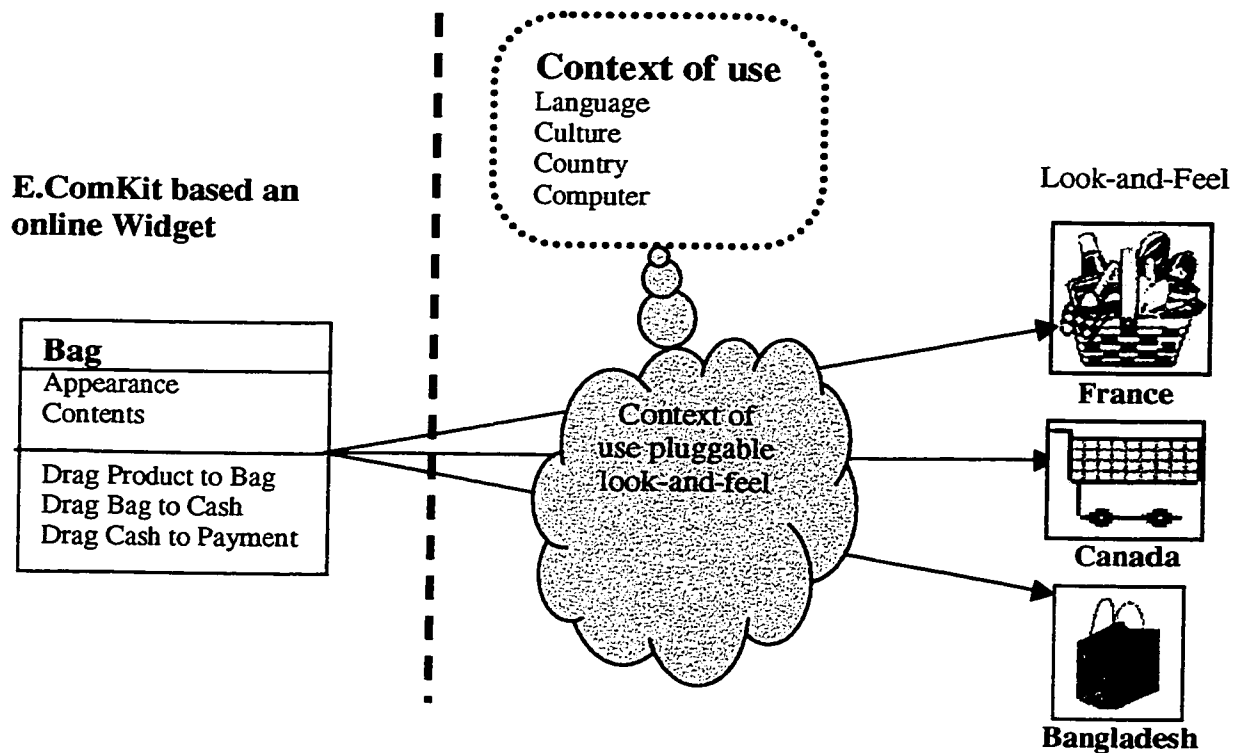


Figure 3: Context of use pluggable look-and-feel examples

Chapter-3

3. An online eBookStore Prototype

3.1 Introduction

Generally, the current online eBookStore (such as Amazon.com) is developed based on traditional GUI for an online shopping (i.e. menus, scrollbars, buttons, and text-input field etc). Which are inappropriate because it has several GUI limitations, and it is very difficult to incorporate in traditional GUI appearance as per user's attitude, user stereotype, and context of use pluggable look and feel and so on.

A RealThing based an online eBookStore (such as E.ComKit based eBookStore) is an interactive interface, which are able to responds to a sequence of users actions (or/ interaction). And the interface is devoted to a large list of efforts to process user actions (or/ interaction). Therefore, the E.ComKit based eBookStore prototype interface is representing the user attitude, behavior, and tasks, those who wants to buy Books, Articles etc over the Internet.

3.2 Design layout of an online eBookStore

An eBookStore is an online virtual shopping mall. The layout of an online eBookStore is 640 pixels by 480 pixels, which fit all standards type display screen for displaying interface appropriately. In E.ComKit based eBookStore prototype interface, the user can interact with the interface, and buy books, articles etc over the Internet easily using any standard size of

computer display screen. Here are the design layouts the user can display product in a window by using mouse click to a product's shelf, and the user can buy product(s) by using drag and drop mechanism with mouse pointer.

- By clicking a Promotion shelf, or/ User shelf, or/ any shelf of products to display product(s) in a window. (SubTable-5 of Figure-4)
- By clicking a Shelf of shelves to display the products shelves in a small layout window as per domain subject area (SubTable-4 of Figure-4)
- By using product drag and drop mechanism, the user can select product(s) and add to the bag or/ user shelf for buying.
- For buying any product, the user drags and drops into a bag, or/ user shelf, or/ product into a cash, drags and drops invoice into a payment and so on.

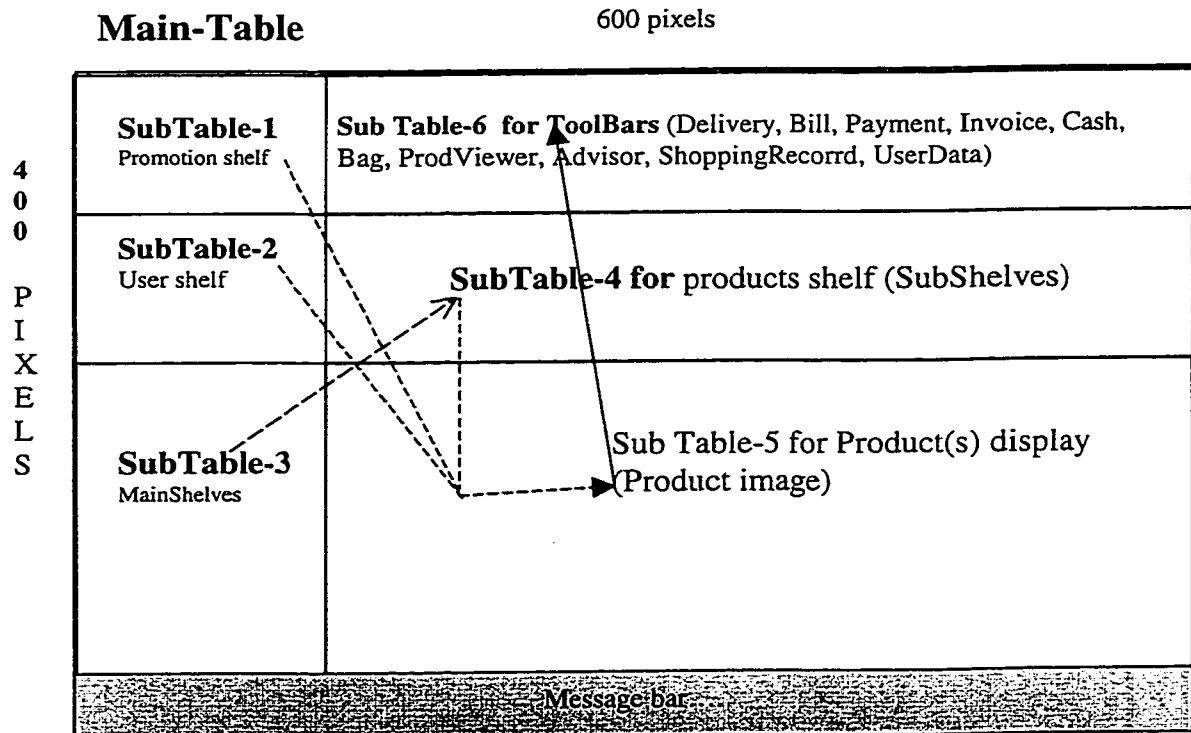


Figure- 4: Design layout of an eBookStore Shopping Mall

3.3 E.ComKit based user interface of an online eBookStore

Figure 5 shows the main user interface for an online eBookStore prototype interface that we have developed using E.ComKit based design approach. This early prototype was developed using HTML, DHTML, Java applets, and JavaScript. And later, we have developed this eBookStore prototype by using Java Development Kits (JDK) including Java Swing Components.

This eBookStore prototype has been developed followed by user preferences and satisfaction, which helps the user to interact it, as per easy-of-use, easy-of-learn, and easy-of-adapt etc.

The eBookStore prototype interface is included RealThing widgets such as shelf, product, user data, shopping record, product viewer, bag, cash, payment, and delivery etc. And the user using mouse pointer, drag and drop mechanism etc for doing all interactions over an online eBookStore site. Such as to click a product shelf to display product in a window, and drag and drop product into a bag or/ user shelf for buying.

For changing a cultural pluggable look and feel of RealThing widget (interface object), move the mouse pointer over the RealThing widget (such as bag, cash etc). And click on right button of mouse to display a list of widgets related to the property of current interface object and as per cultural pluggable look and feel. Then the user can choose and select to display, and change the current RealThing based widget on the eBookStore interface as the user can satisfy his preferences and cultural pluggable look and feel.

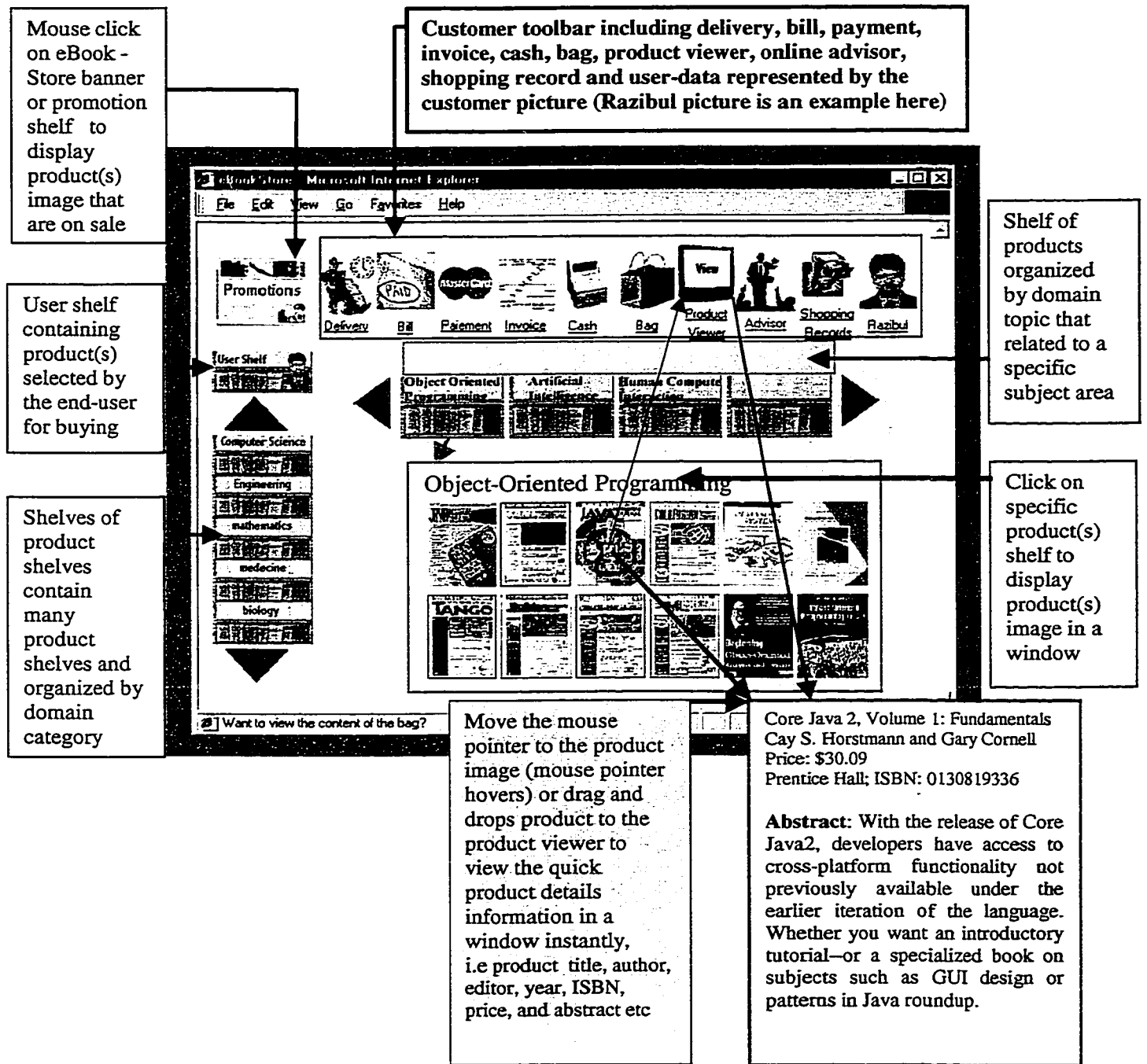


Figure 5: E.ComKit based an online eBookStore

3.4 Traditional GUI based User Interface an online eBookStore

Figure 6 shows the main user interface for an online eBookStore that has been developed using traditional GUI based design approach such as Amazon.Com. And it is very difficult to incorporate user attitude, stereotype, and context of use pluggable look and feel to the traditional GUI based web application.

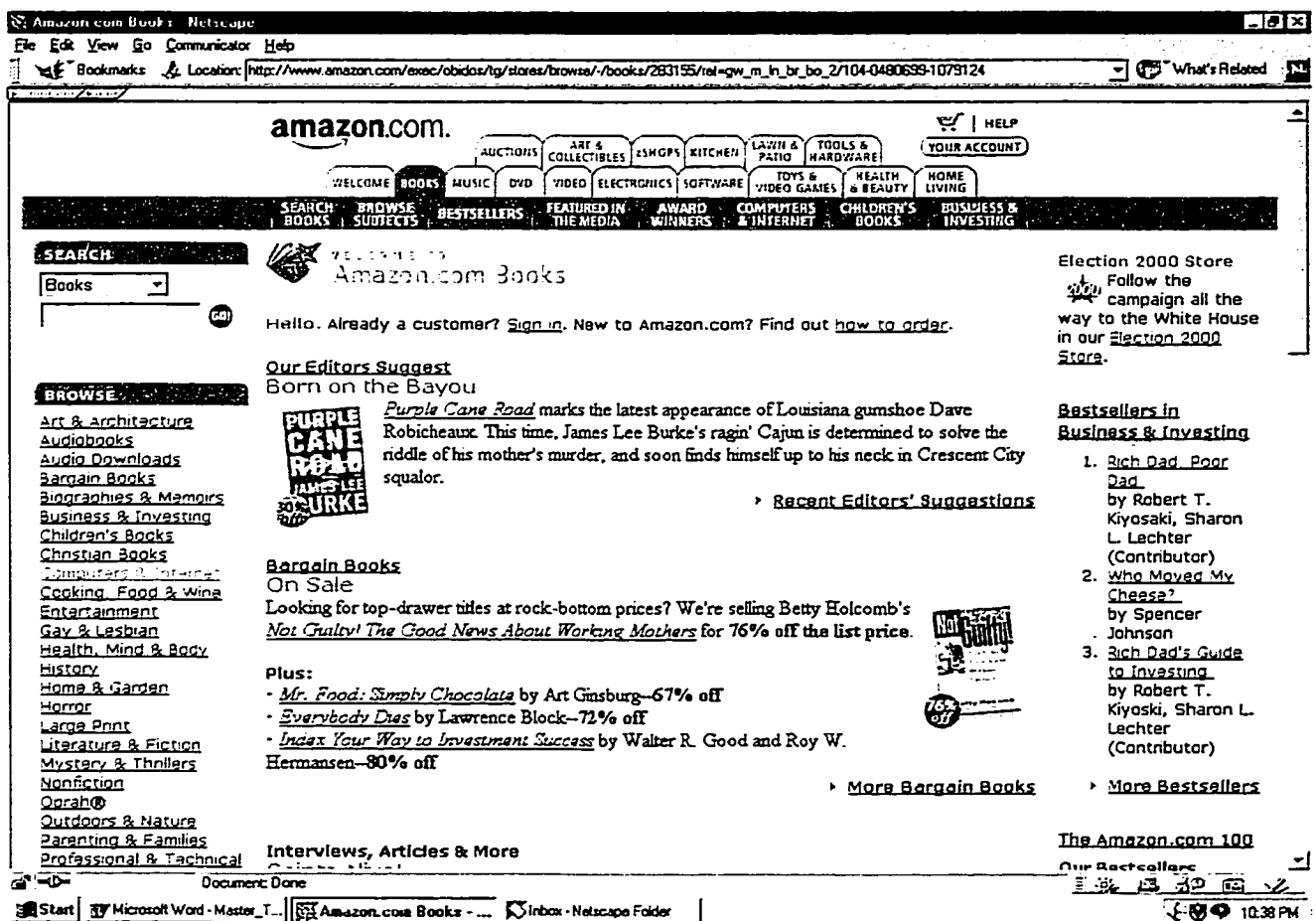


Figure-6: Traditional GUI based an online eBookstore such as Amazon.com

3.5 Evaluations of Traditional GUI based, and E.ComKit based eBookStores according to Norman Principles

The evaluations have been done between traditional based Amazon.com eBookStore (Figurer-6) and E.ComKit based eBookStore (Figure-5) as conformance of Norman Principles (Table-3).

Norman's Principles	Traditional based eBookStore Such as Amazon.com	E.ComKit Based eBookStore
<p>Affordance: Provide indications of making understanding easily by the user. i.e. What to do next? How to use it?</p>	<p><i>The perceived and external properties are traditional GUI based interfaces, and are not determined easily by the user that how the object is used?</i></p> <p><i>It does not match the RealThing based RealWorld behavior, and it has not a good affordance that can be understands easily by the user.</i></p> <p><i>It needs to changes pages to find detail description of interface behaviors and action.</i></p>	<p><i>The perceived and external properties are E.ComKit RealThing based widgets and are determined easily that how the object(s) is used based on its visual look and feel.</i></p> <p><i>RealThing based eBookStore widgets does represent good affordance for understanding by the user, because its look and feel matches the real world object. It does not need to know the detail description of behaviors for an action.</i></p>

Norman's Principles	Traditional based eBookStore Such as Amazon.com	E.ComKit Based EbookStore
<p>Mapping: <i>Something that can be measurable and understandable based on the set of possible relations between objects.</i></p>	<p><i>In traditional based GUI user interface such as Amazon.com does not relate 100% RealWorld behaviour to keeps the same status and functionality in all pages, if its change pages. To many information in every page and difficult to determine a good measurable mapping status. It is not a good measurable and understandable by the user easily, because it has no natural relationships between GUI objects and RealWorld objects.</i></p>	<p><i>In eBookStore all RealThing based user interface widgets are dynamic, but it keeps the same status and functionality, though its changes the look and feel of pages, and it matches the same behaviour like RealWorld objects. It is measurable and understandable by the user easily because it has natural relationships between eBookStore widgets and RealWorld objects.</i></p>

Norman's Principles	Traditional based eBookStore Such as Amazon.com	E.ComKit Based EbookStore
<p>Conceptual Model: It is implicated to understand the model. How the operating parts work or/ how objects behave or/ how functionality works.</p>	<p>The Amazon.com is designed to develop a GUI Interface as per traditional GUI based design approach. Which does not refer mental model. The user may difficulties to understand all the GUI interfaces of its behaviour, and functionality.</p>	<p>The eBookStore is designed to develop a RealThing Interface based on E.ComKit design approach, which are refer to the RealThing widgets that can satisfy mental model appropriately. The user may understand easily all the widgets behaviours and functionalities.</p>
<p>Causality: The result happened, or/ occurred right after doing an action, and it provides clear context-oriented and interpretable "feedback" after each action.</p>	<p>By clicking a GUI interface its change the page, and lost or/ hide the previous page. Then the user needs to remember something to take an action on the next page otherwise the user will lost completely.</p> <p>Remembering information by the user is not good causality.</p>	<p>What happens immediately following an action in E.ComKit based eBookStore is to be caused that, as soon as the user click any RealThing interface, it will view the information in the same page. Or if the mouse pointer places a product image into a bag, just its changes the appearance of Bag in the same page and change from empty bag state to non-empty bag state etc.</p>

Norman's Principles	Traditional based eBookStore Such as Amazon.com	E.ComKit Based EbookStore
<p>Visible Constraints: It is visible and measurable for one direction.</p>	<p>The actions should be perceived from the GUI interfaces that the user does not able to make interfaces visible, because every click or/ an action, it change the pages. In the most case the visible constraint is unknown.</p>	<p>The actions should be perceived from the RealThing widget's look-and-feel that the user is able to make the interface visible because every click or/ an action, it does not change pages, and it remain in the same page. Constraints are known in most cases.</p>
<p>Population Stereotypes: Learn idioms that work in a certain way will satisfy every user's culture and social stereotypes.</p>	<p>All idioms of GUI interface works in different way. In most cases it does not refer the real world environment, and does not change interface look and feel based on context of use (i.e. as per culture and social dimension), and preferences.</p>	<p>All idioms associated with E.ComKit widget work in same way that they do act in real world environment. In RealWorld different culture and society are acted in different way for each and every possible action. The eBookStore RealThing widget changes its look and feel based on culture and social dimension. Because it has natural relationship between RealThing based widget and RealWorld object.</p>

Norman's Principles	Traditional based eBookStore Such as Amazon.com	E.ComKit Based EbookStore
<p>Transfer effect: Has both learning and expectations, i.e. based on</p> <ul style="list-style-type: none"> - Positive transfer or - Negative transfer 	<p><i>It acts some times as positive transfer effects, and some times negative transfer effects that can be learned by using mouse pointer, and by doing interaction. And it needs to learn some functionalities prior to use, because in most cases the user does not know what is happening in next step.</i></p>	<p><i>The eBookStore has positive transfer effect, learning all by using mouse pointer and interaction. But does not need to learn any functionality prior to use it, because the user already know the RealWorld based RealThing behaviour.</i></p>

Table-3: Evaluation of Amazon.com, and E.ComKit based eBookStore

3.6 Feedback and comments after evaluating both eBookStore interfaces

After examination of both eBookStores interfaces (“the Amazon.com, and E.ComKit based eBookStore interfaces”), we find differences (Table-3) as per **Norman's Principles** to satisfy the user preferences that how easy to use, and how easy to interact, and so on.

These investigation helps to determine, whether the user understand easily or/ not by interacting web interface, that how to use it, and how to interact with efficiently and effectively to get a useful results.

If the user perceives the RealThing based user interface components, or/ the traditional based user interface components, and or/ to find out if there is anything offensive in the product.

The traditional based an online eBookStore user interface look and feel does not satisfy the RealThing based design (such as Amazon.com), and does not widely understood interface elements (i.e. Image, icons, and windows) to interact over the Internet.

But the E.ComKit based an online eBookStore user interface look and feel to satisfy the RealThing design, and widely understood interface elements (i.e. Image, icons, and windows) to interact over the Internet. The user can find a specific product from a large list of products and obtain information about it quickly and easily. And the user can accomplishes all these tasks just by using mouse pointer with dragging and dropping mechanism.

In complex shopping tasks in E.ComKit based an online eBookStore, such as checkout and delivery are more natural and intuitive. The users can customize the shopping mall by creating own interface (as example of user-shelf), and or/ by redesigning, and or/ by reorganizing existing widgets, and or/ changing pluggable look and feel of an interface object (i.e. widget) as per context of use and user stereotypes. And the user can re-used or/ interacts the component-based E.ComKit widgets by unlimited number of times in virtual shopping environment.

Chapter-4

4. E.ComKit based eBookStore OO Modeling

4.1 Introduction

Developing a model for an industrial-strength software system prior to its construction or/ renovation is as essential as having a blueprint for building [OMG-news]. The Object Oriented Modeling is an integrated approach to analysis, design and implementation. It provides a novel method to solve software problems using models organized around the Real-World concepts. And it is a technique based on modular decomposition of a system for system manipulates. The OO analysis defines domain problems i.e. to define the Real-World objects and it's associate properties. The objects are required to build the model will be identified, and a system will be developed that its simulates the Real-World model. Each of the object will be acted independently and managed its own interaction and data. And one or more objects needed to be included in the system to manage its interface. The model allows the user to specify how many object interactions should include to a given run. The E.ComKit based an online eBookStore Modeling represents all objects that are needed to analysis, design and implementation of an online eBookStore over the Internet.

4.2 Component-based E.ComKit RealThings User Interface:

Component-based E.ComKit RealThings user interface is providing the advantages to re-use widgets in the virtual shopping environment. (i.e. Bag, Cash etc) and it can be re-used by an unlimited number of interactions. A component-based E.ComKit RealThings user interface is a collection of widget(s) that are used to interact by the user and a realization of a set of widgets in general. All components are task-oriented classes in E.ComKit for any eCommerce virtual shopping or/ web environment i.e. A Bag is a container that contains list of product(s) for buying. Cash is a widget to count product(s) and cumulates prices to generate an invoice. An Invoice is a widget that contains products tile, price, cumulative total price and so on.

4.3 The E.ComKit Modeling for an online eBookStore

OO modeling approach helps to understand the overall system interactions and functionality, and typically the system is specifying the object(s) to produce another object(s) followed by user interaction. In the following structure of E.ComKit model, it describes the site map of an object(s), and interaction(s) to display product image, which is book image compared to the real-world object as per user cultural pluggable look and feel. The interaction mechanisms of an online eBookStore are more natural and intuitive, because its use mouse pointer and drag and drop mechanism for an interaction to the RealThing object i.e. MouseOver, MouseClick, Mouse Drag and Mouse Drop and so on.

4.4 E.ComKit Object-Orientation

A class is set of objects that share the same attributes, methods, relationships and semantics. A class is rendered as a rectangle usually including class identifier, which is name of class, attributes, and operations that could implement interface object. And an interface is a collection of operations that specify a service of class or/ component. The Object Oriented model records the entities and their relationships. The entities are organized into classes. It includes physical entities and concepts.

In the E.ComKit based an online eBookStore Object Oriented Classes are the Shelf, Promotion Shelf, User Shelf, Product Viewer, Bag, Cash, Payment, and Delivery etc. Each of individuals is working independently but may have relationship with other classes.

4.4.1 Assumptions

It assumed that the each shelf contains many shelves (or/ products). And it will be display the shelf of product(s) [or product(s)] at the user's site right after interacts (click) on it by mouse pointer. So we can exclude the concurrency between interacts or/ request due to the specific characteristics of an online application for each user. Because there is only one mouse pointer to click on the interface object for one specific user, and only one input event will be generated at a time.

4.4.2 Generalization and Specialization

A shelf of shelves, or/- promotion shelf, or/-user shelf, and or/-bag is a part of shelf. So a shelf is a generalization of an online eBookStore application. Or a shelf of shelves, or/ promotion shelf, or/ user shelf, and or/ bag is a specialization of shelf.

A shelf is an open or/ a close state, if opened state then an interaction has been done by a mouse pointer on it for displaying shelves of product(s) [or/ product(s)]. Or if the shelf is a close state that means the interaction has not been done yet.

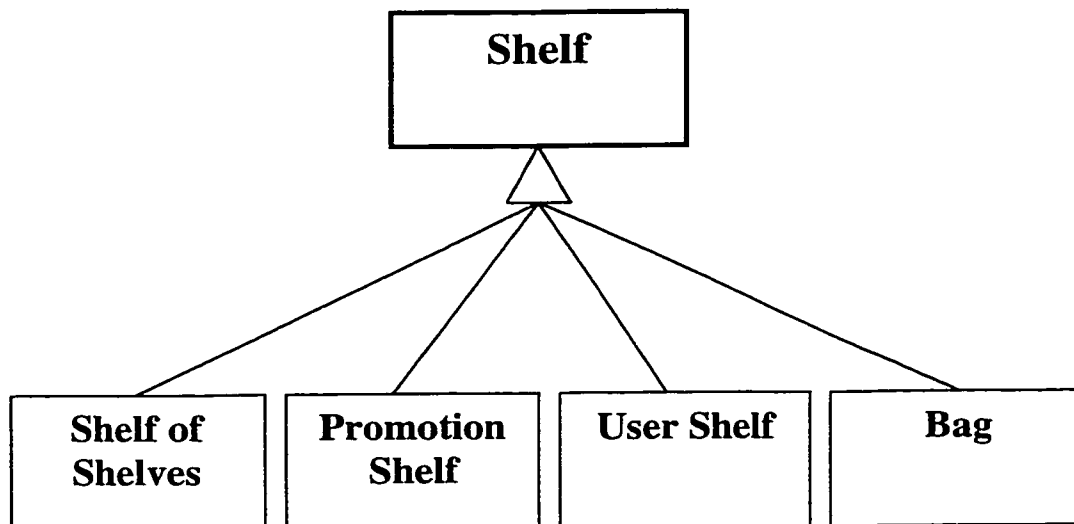


Figure-7: Shelf is a generalization

4.4.3 Product to Cash

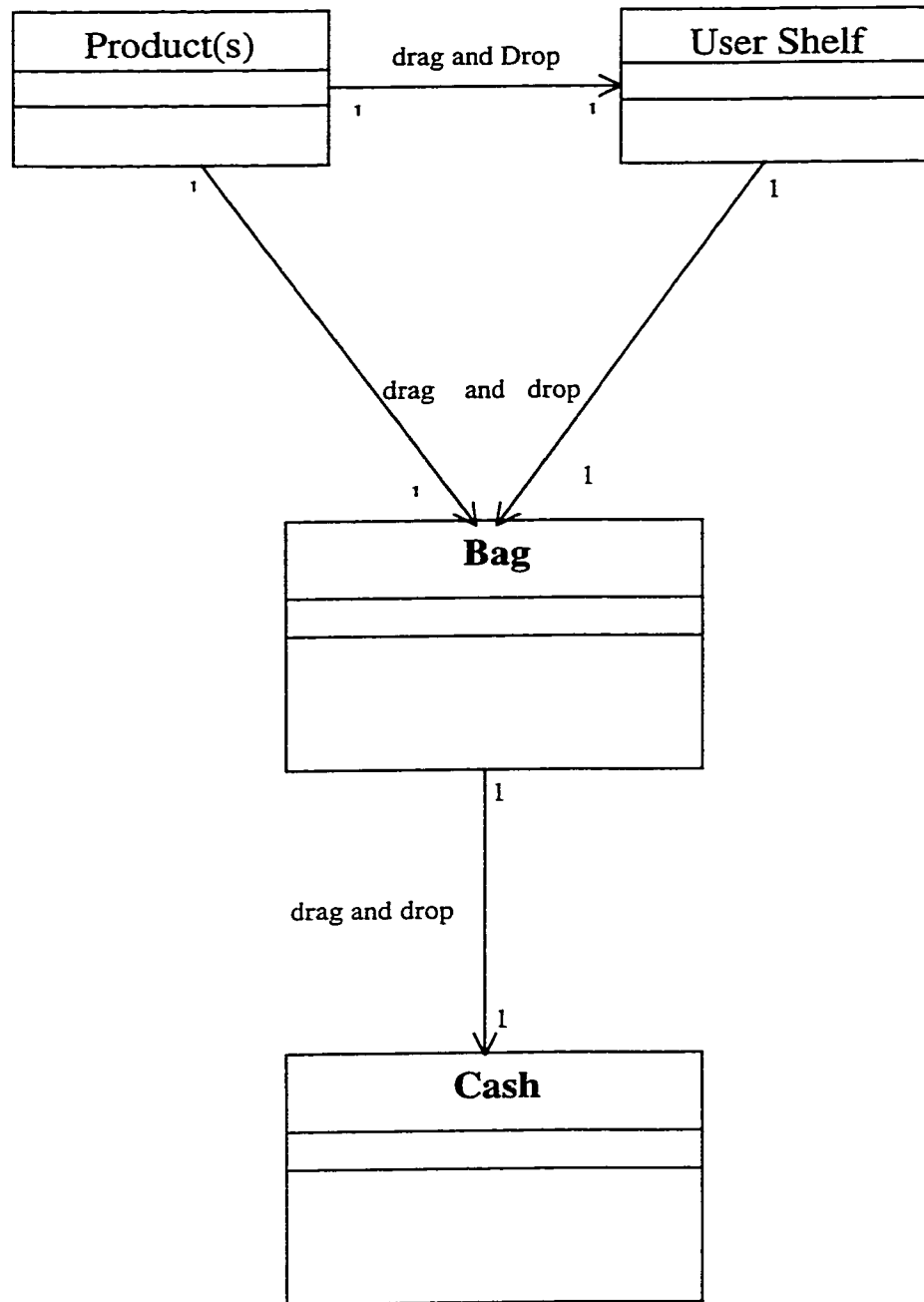


Figure-8 : Product to Cash

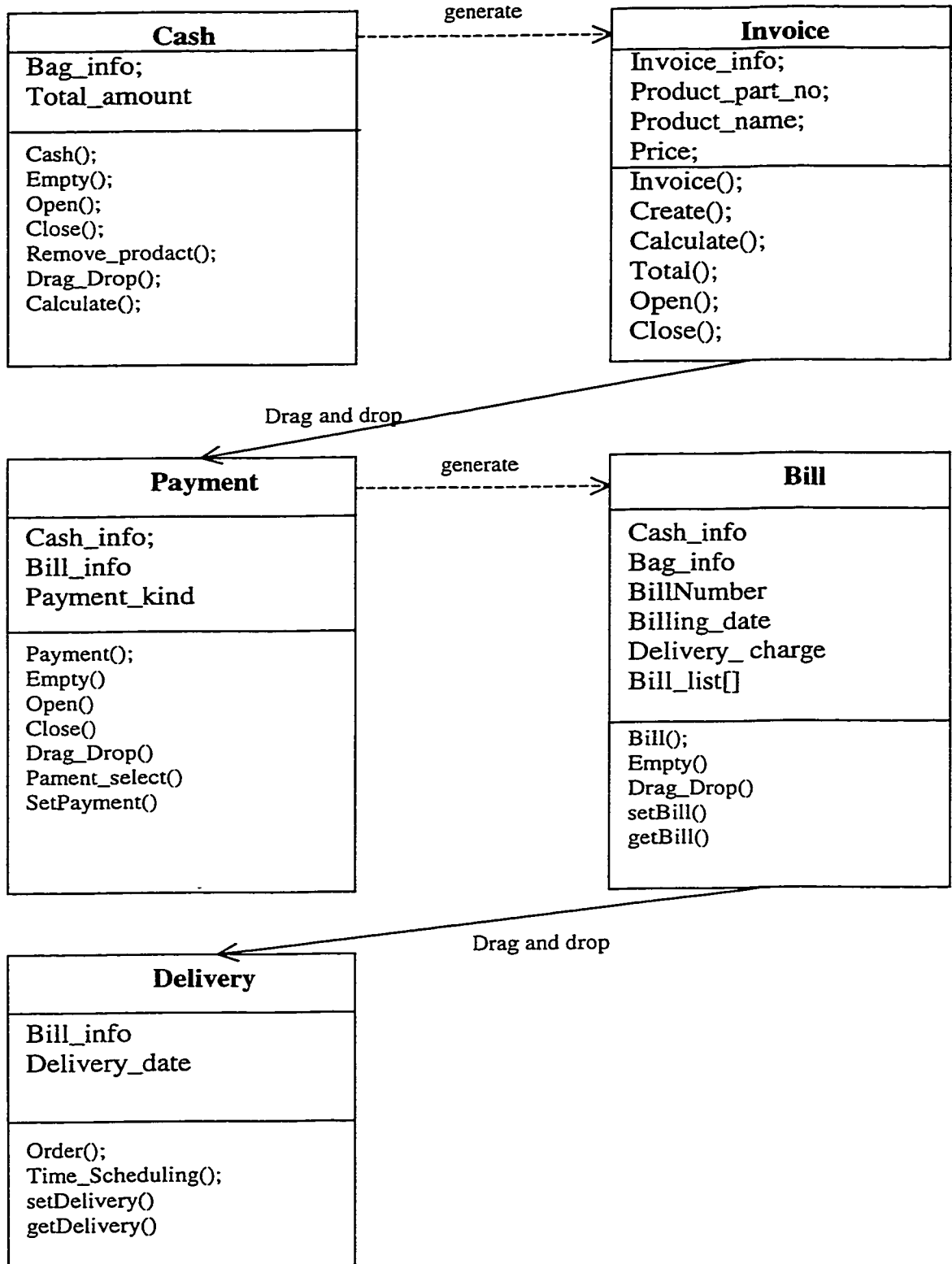
4.4.4 E.ComKit based an eBookStore Class Details

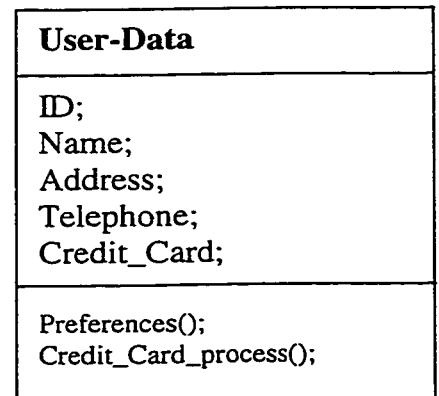
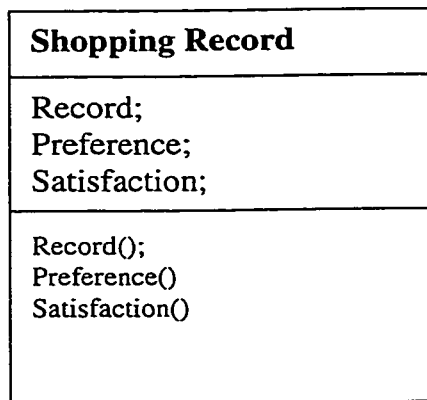
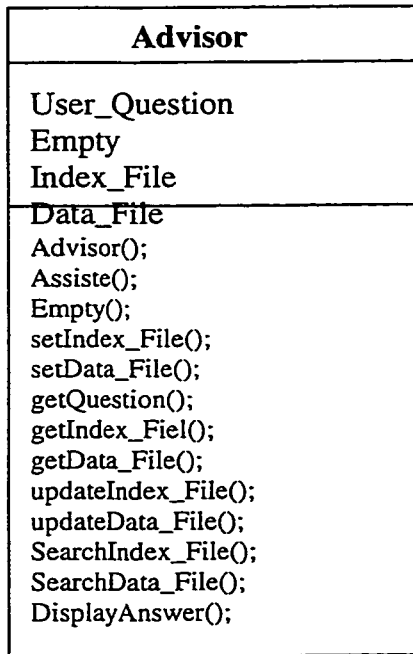
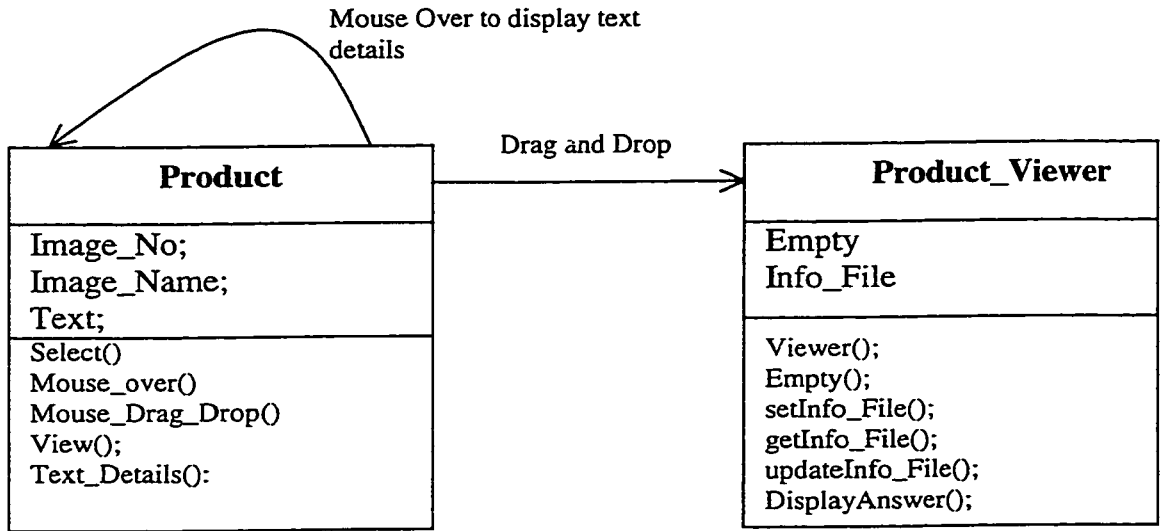
Shelf
Look_Feel Images_List[] Color Product_List Product_group
Shelf() //Constructor addImages(); setLook(); setColor(); getImages(); Select_Group(); Change_look(); Change_Color(); Add_New_GroupProduct(); Remove_GropeProduct(); Open_Product_List();

UserShelf
List_Product[] User_name User_info Date_Purchase[]
UserShelf(); //Constructor Empty(); getImage(); getInfo(); getDate(); setImage(); setInfo(); setDate() Open_List();

PromotionShelf
ListProduct[] Start Finish Special_Price[]
PromotionShelf(); setStart(); setFinish(); setSpecialPrice(); getStart(); getFinish(); getSpecialPrice(); Empty(); Add();

Bag
Empty Total_Amount Payment
Bag() //Constructor Empty() Add() Open() Remove() Close() Delete_all() Move() Drag_Drop() Calculate()



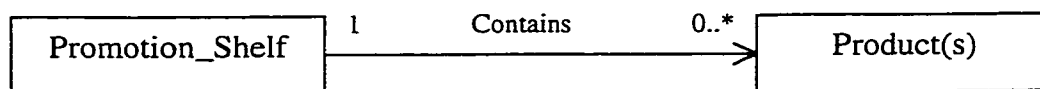


4.5 E.ComKit based an eBookStore library and relationship between classes

A typical E.ComKit toolkit “an eBookStore object oriented library” is included interactive object oriented classes and relationships between classes are as follows:

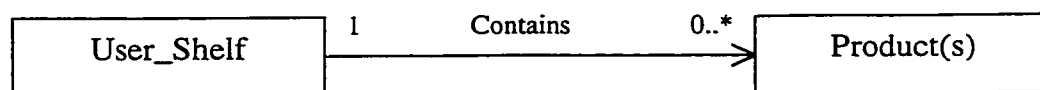
1. Promotion shelf [Product(s) Shelf]

The promotion shelf contains zero or many products in it, when interact on it by the mouse pointer, its display product images into a window, which are on sale or/ promotion of products.



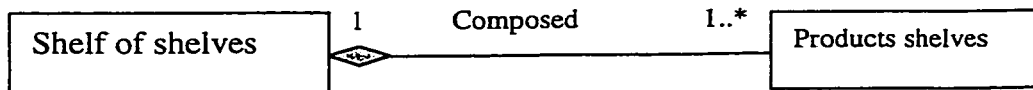
2. User Shelf [Product(s) Shelf]

The User shelf contains zero or many products in it, when interact on it by the mouse pointer, its display product images into a window. Which are selected by the user for buying and drag product(s) from the product's window of different shelves, and drop into the User shelf.



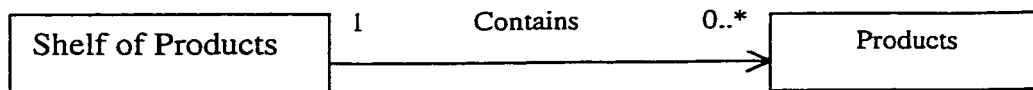
3. Shelf of shelves (Main Shelves)

A shelf of shelves is related to one or many product shelves. i.e. The Shelf of Shelves composed of one or many shelves, when interact on it by the mouse pointer, its display all products shelves into a panel window.

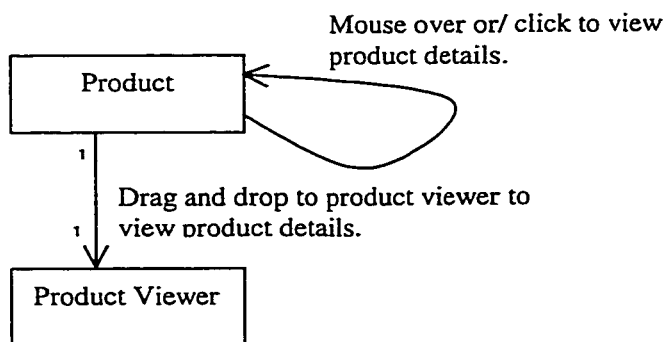


4. Shelf of Product [Sub shelves, Products shelf]

The shelf of product contains zero or many products in it, when interact on it by the mouse pointer, its display products images into a window.

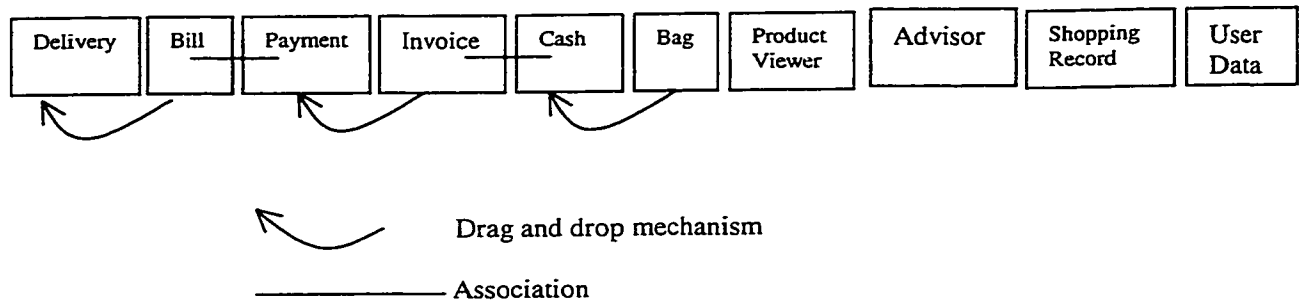


5. Product to display details information in a window. The user can move mouse pointer over the product image or/ drag and drop product image to product viewer to display product details in a window.

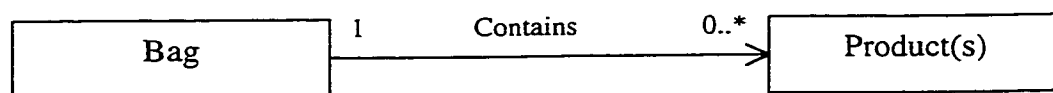


6. Interactive ToolBars (Interactive container)

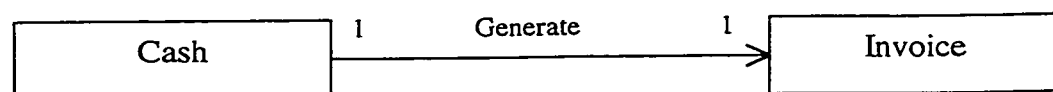
A toolbar is an interactive container, which contains many interface objects. All interface objects are the RealThing based widgets.



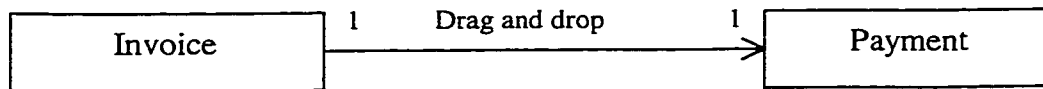
6.1 Bag: A bag is a temporary shelf that it contains a zero or many list of products that the user can store product(s) for buying using drag and drop mechanism. Every time the product is added to the bag, its update a list of products automatically. It displays a list of products, when the mouse clicks on it.



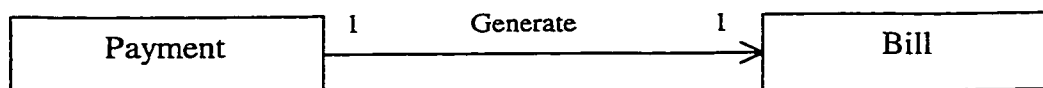
6.2 Cash: A cash generates an invoice, when product(s) is drag and drop into a cash, it's generated an invoice automatically.



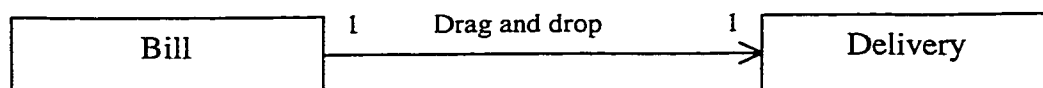
6.3 Invoice: An invoice is a list of products that are being processed for buying, the invoice needs to drag and drop into a payment for paying.



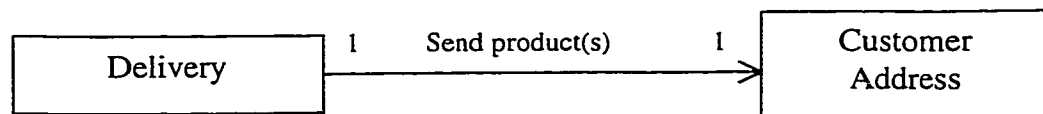
6.4 Payment: A payment accepts in different way of payment (i.e. credit card, eCash etc payment) and generates a bill, when payment is done, it's generated a bill automatically.



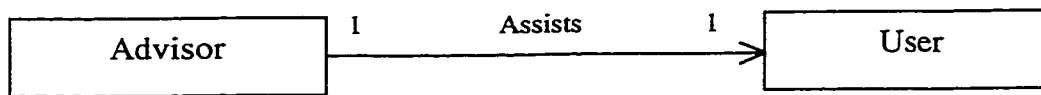
6.5 Bill: A bill is a receipt of payment, the bill needs to drag and drop into a delivery to deliver product(s) to the customer's address.



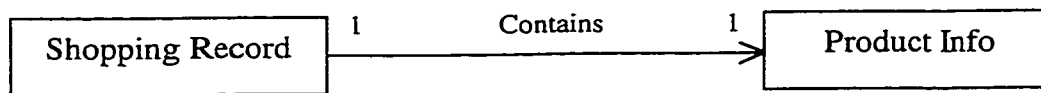
6.6 Delivery: A delivery arranges to delivery a product(s) as per specific time scheduling, it includes customer delivery address to delivery a product(s).



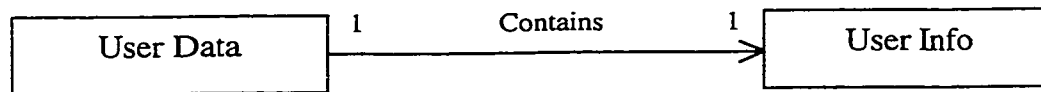
6.7 Advisor: An advisor is an online agent, who assists user over the Internet, if the use needs any.



6.8 Shopping Record: A shopping record is a list of product(s), or/ user preferences, in which the user already has bought product(s) over the Internet.



6.9 User Data: A user data contains all personal information of user, in which the user needs to use it to buy and delivery product(s) over an Internet.



4.6 An eBookStore Object Oriented Class Diagram

Class diagram is the backbone of nearly all Object Oriented modeling, and it comprises the maximum efforts of modeling the Object Oriented system. It describes the types of objects in the system that includes a set of interface objects. And its collaborations and their relationships to address the static design view of an online eBookStore that shows a collection of declarative

objects. The E.ComKit based eBookStore OO class diagram (Figure-9) presents a set of RealThing OO classes that are invoked independently right after interact by the mouse pointer.

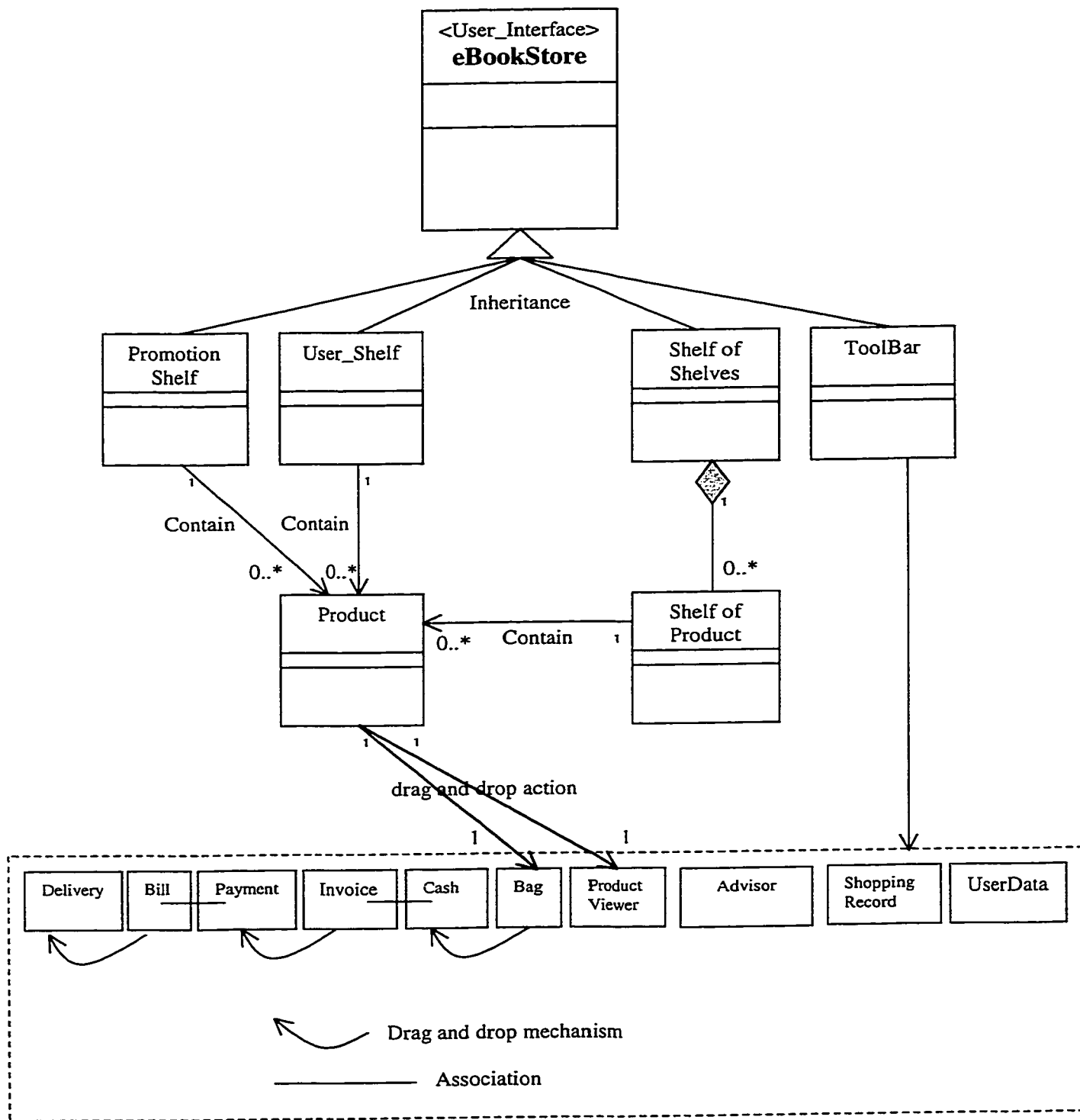


Figure- 9 : An eBookStore Object Oriented Class Diagram

Chapter-5

5. Lessons Learned and Perspectives

5.1 Conclusion and future work

In this work, we have presented and outlined a toolkit for developing a new generation of user interfaces for eCommerce or/web applications. It is an alternative approach to current GUI and HTML-based web designs approaches. The E.ComKit introduces a list of real-world widgets whose look-and-feel (widget) varies according to the culture and user stereotype. Each widget delegates the look-and-feel-specific aspects of its responsibilities to whatever context of use is currently installed. The context of use includes information about the user behavioral, tasks, as well as the technical (operating system and hardware), social and cultural environments.

The evaluations and preliminary tests we carried out have led us to enhance E.ComKit initial approach and to complete its specification. Another key direction in our work is to extend E.ComKit to support development of multiple user interfaces, particularly for personal digital assistants (PDA) for an eCommerce or/ web application.

Examples of look-and-feel were provided throughout this research work to further illustrate this principle. Users and developers alike must define the look-and-feel to represent the best culture and stereotype of future users. The look-and-feel of the different widgets should be developed according to the example of Norman's principles that make the design cognitively respectable [3].

5.1.1 Customization of an online shopping environments

For customization of virtual shopping mall, the interface look and feel need to be changed dynamically according to the context of use, and user preferences. The context of use is dependable to the culture, society, country, and geographical area etc. The RealThing based E.ComKit approach will be useful to customize an online shopping environment and it can satisfy the needs of user, especially in the context of international audience.

Therefore, it becomes a necessity that users can redesign, and or/ reorganize, and or/ create their own interface. It is an important reason why the World Wide Web will be so successful, because everyone can redesign, or/ reorganize, or/ create his/her own interface on the web as per preferences by using E.ComKit approach.

Developing user interface by using E.ComKit design approach, allows users to accomplish the following functionalities on the fly:

- Change the look-and-feel of interface widget as per user preferences.
- Customize shelf of product(s) by dragging and dropping products from existing shelves to a user shelf, which he or/ she can create as per preference.

- Create new widgets. For example, a customer wishes to create a bag and payment for a friend, who wants to make shopping over the Internet. To make a payment, the user right-clicks and selects *Create Payment* from the contextual menu.
- A new payment widget will appear. The customer enters the necessary information (i.e. card number, expiry date and name of cardholder) to Iconize for payment.

In E.ComKit approach, the user can also design interface over the internet, is the part of the underlying toolkit, as recommended by Myers [1] and needs to be implemented in user design and programming environments [5].

5.1.2 Changing Look and Feel of RealThing Interface Object

The RealThing widget look and feel can vary according to context of use and user stereotype. If the user redesigns his or/ her own interfaces on the fly, then the RealThing based interface look and feel will be changed automatically.

The user can add a new property to the context of use as per user stereotype, if needed. The User Design, RealThing approach, and Context of use are highly coupled and co-related modules (Figure-10). If anything is changed as per user preferences and satisfaction from any part of the three modules, i.e. User Design, RealRhing approach, and Context of use, the other will be updated accordingly. By changing anything from any module, the other two modules will be changed automatically.

So, the User Design, RealThing approach, and Context of use can be customized by developping its E.ComKit approach, and or/ changing look and feel of RealThing interface object.

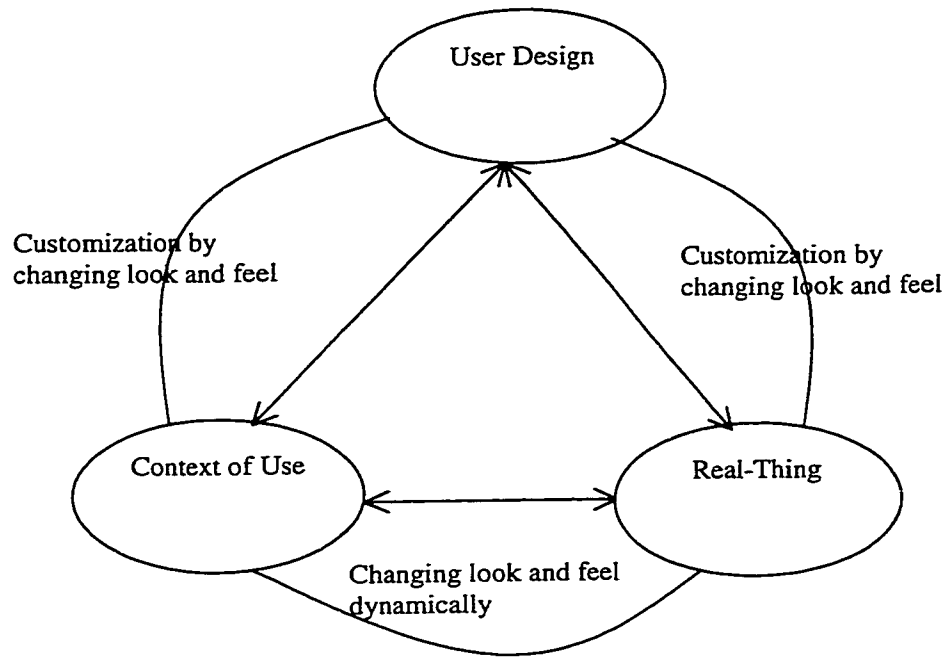


Figure-10: Changing look and feel of RealThing objects

5.2 Discussions and Achievements

A recent report released by Creative Good estimates in 1999 [17] that \$6 billion was lost due to poor web experiences. As per usability the web application needs to be redesigned to meet the user preferences and satisfaction i.e. the web application needs to be focused on ease of use, ease of learn and ease to adapt. For instance, the importance of usability was proved when IBM gathered experience at 400% increase of sales, right after redesigning the website as per usability in mind. Presently, ease of use is a critical important issue to get more users interact on the eCommerce or/ web application.

The RealThing based widgets of any web, or/ eCommerce shopping mall are a list of Usability templates in E.ComKit as per end-user view. And a RealThing-based E.ComKit UI approach brings distinction between novice, causal and expert users, and takes into account both users to provide usability guideline with effectively and efficiently. The quantitative and qualitative achievements of E.ComKit RealThing widgets are easy-to-learn, easy-to-adapt and easy-to-use by both users. And the qualitative and quantitative achievements of E.ComKit RealThing widgets are the satisfaction with a certain number of percentage from 0 to 10 dependable on the type of users, i.e. effectiveness, efficiency, enjoyability, simplicity, consistency.

In general E.ComKit RealThing approaches have the high level objectives to reach the goal or/ quality to satisfy the user requirements in respect to efficiency, effectiveness, learn-ability, and adaptability and so on.

Appendix-A

1. E.ComKit based of an eBookStore Model

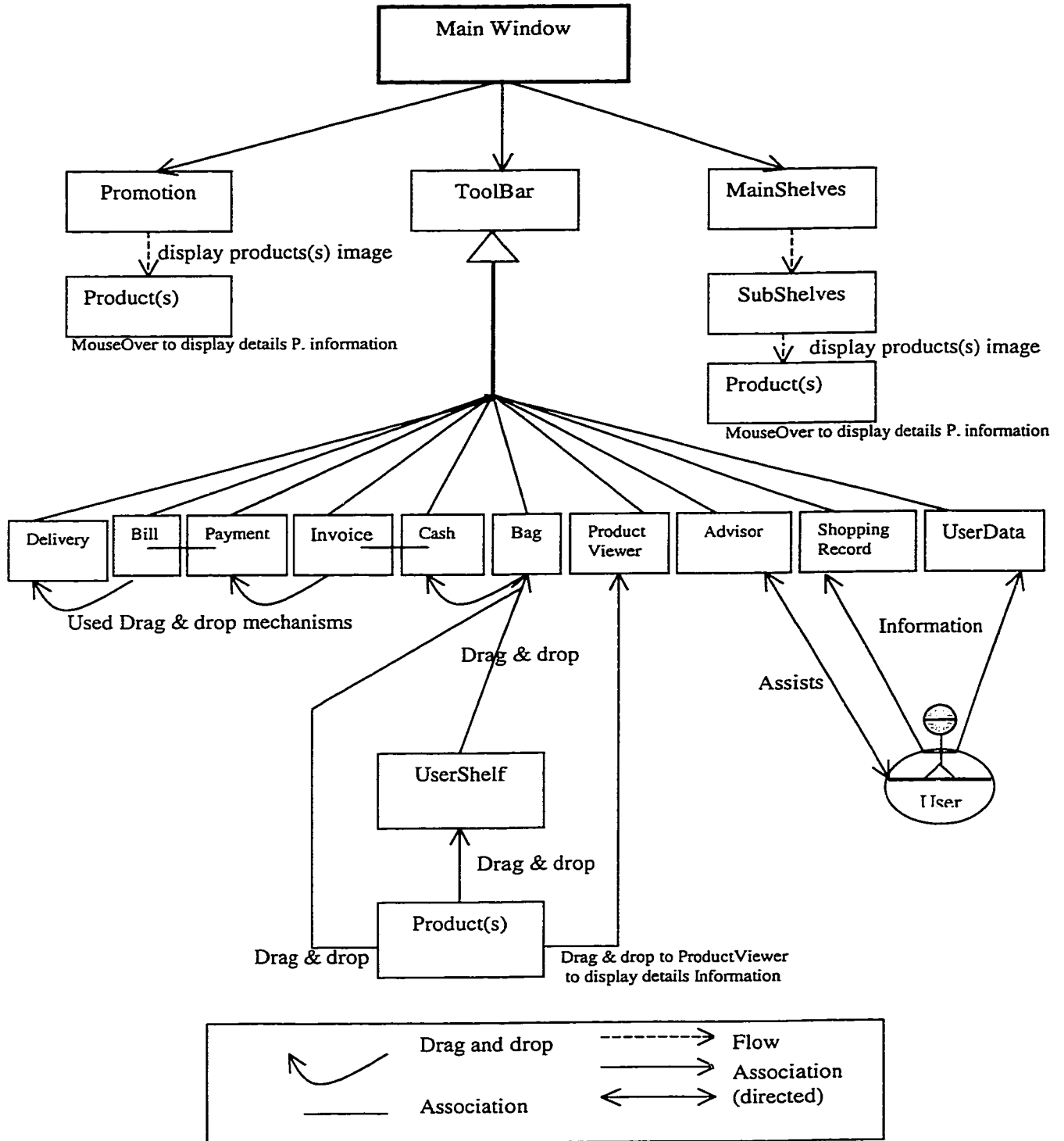


Figure- 11: E.ComKit based eBookStore Model

2. Each Widget is an object of an eBookStore Model:

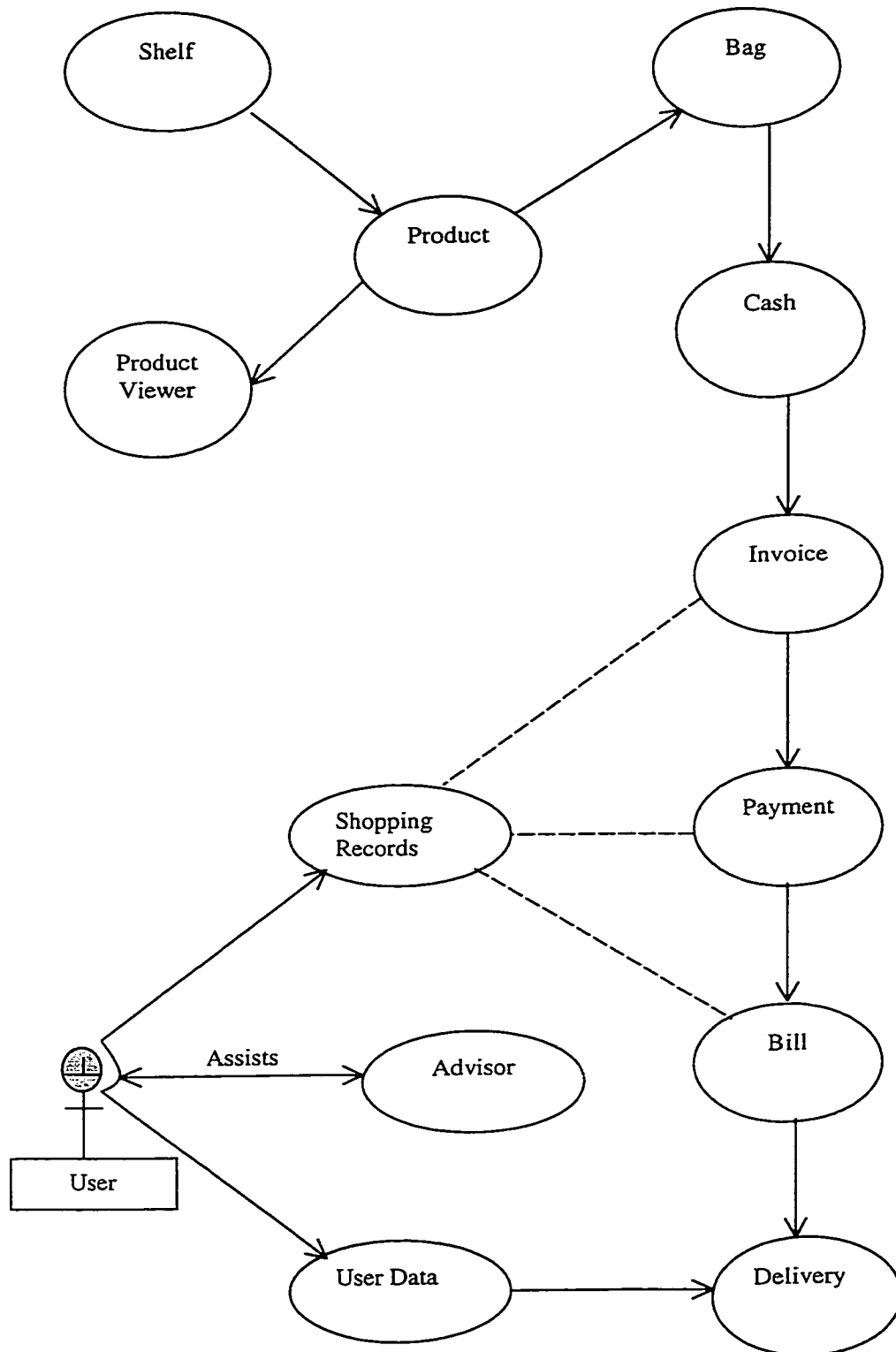


Figure- 12 : List of objects of eBookStore Model

Appendix-B

1. Java source codes (as sample examples) of an online eBookStore:

1.1. eBookStore.java

```
// RealThing based eBookStore prototype Interface
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;

public class eBookStore extends JFrame{
    JScrollPane scrollPane = new JScrollPane();
    JTextField status;
    TableOne tOne;
    ButtonTableTwo tTwo;
    TableThree tThree;
    TableFour tFour;
    TableFiveSix tSix;
    TableSeven tSeven;
    BarOne barOne;
    BarTwo barTwo;
    JPanel buttonPanel;
    JButton back;
    JButton next;

    public eBookStore() {                                // Constructor
        setTitle("EBookStore");
        setResizable(true);
        setSize(600,400);
        setFont(new Font("SansSerif",Font.BOLD,14));
        setBackground(SystemColor.window);
        addWindowListener(new WindowAdapter()
        {

            public void windowClosing(WindowEvent e){System.exit(0);}
        });
        Toolkit tk = Toolkit.getDefaultToolkit();
        Dimension d = tk.getScreenSize();
        int screenHeight = d.height;
        int screenWidth = d.width;
        setLocation(screenWidth/4,screenHeight/4);
        Container contentPanel = getContentPane();
```

```

Image [] images = {
tk.getImage("Images/abook1.gif"),
tk.getImage("Images/abook2.gif"),
tk.getImage("Images/abook3.gif"),
tk.getImage("Images/abook4.gif"),
tk.getImage("Images/abook5.gif"),
tk.getImage("Images/abook7.gif"),
};

```

```

Image [] face = {
tk.getImage("CsOOPproducts/oop7.gif"),
tk.getImage("CsOOPproducts/oop6.gif"),
tk.getImage("CsOOPproducts/oop5.gif"),
tk.getImage("CsOOPproducts/oop4.gif"),
tk.getImage("CsOOPproducts/oop3.gif"),
tk.getImage("CsOOPproducts/oop2.gif"),
tk.getImage("CsOOPproducts/oop1.gif"),
tk.getImage("CsOOPproducts/oop5.gif"),
tk.getImage("CsOOPproducts/oop7.gif"),
tk.getImage("CsOOPproducts/oop6.gif"),
tk.getImage("CsOOPproducts/oop3.gif"),
tk.getImage("CsOOPproducts/oop2.gif"),
tk.getImage("CsOOPproducts/oop4.gif"),
tk.getImage("CsOOPproducts/oop1.gif"),
tk.getImage("CsOOPproducts/oop4.gif"),
tk.getImage("CsOOPproducts/oop1.gif"),
};

```

```

Image barOneImage = tk.getImage("Images/Tutorial.gif");
Image barTwoImage = tk.getImage("Images/Tutorial.gif");
barOne = new BarOne(barOneImage);
barTwo = new BarTwo(barTwoImage);
tSix = new TableFiveSix(face,this);
tOne = new TableOne("Home_EBook_1.gif",tSix);
tTwo = new ButtonTableTwo();
tFour = new TableFour(images,tSix);
tThree = new TableThree(tFour,barOne,barTwo,tSix);
tSeven = new TableSeven();
Box windowBox = Box.createVerticalBox();
Box bottomBox = Box.createHorizontalBox();
Box firstBox = Box.createHorizontalBox();
Box secondBox = Box.createVerticalBox();
Box thirdBox = Box.createVerticalBox();

```

```

firstBox.add(tOne);
firstBox.add(tTwo);
secondBox.add(tSeven);
secondBox.add(tThree);
thirdBox.add(barTwo);
thirdBox.add(tFour);
thirdBox.add(tSix);
bottomBox.add(secondBox);
bottomBox.add(barOne);
bottomBox.add(thirdBox);
windowBox.add(firstBox);
//windowBox.add(Box.createVerticalStrut(10));
windowBox.add(bottomBox);
JPanel statusPanel = new JPanel();
contentPanel.add(statusPanel, BorderLayout.SOUTH);
scrollPane = new JScrollPane(windowBox);
contentPanel.add(scrollPane, "Center");
}

public static void main(String[] argv){
    eBookStore testFrame = new eBookStore();
    testFrame.show();
}
}

```

1.2 TableOne.java

```

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;
import java.io.*;

public class TableOne extends JPanel implements MouseMotionListener{
    JButton button = null;
    JPanel buttonPanel;
    Icon icon = null;
    Border buttonBorder;
    boolean in;
    TableFiveSix picArea;
    Toolkit tk = Toolkit.getDefaultToolkit();
    Image [] str = {

```

```

tk.getImage("PromotionProducts/Promo1.jpg"),
tk.getImage("PromotionProducts/Promo2.jpg"),
tk.getImage("PromotionProducts/Promo3.jpg"),
tk.getImage("PromotionProducts/Promo4.jpg"),
tk.getImage("PromotionProducts/Promo5.jpg"),
tk.getImage("PromotionProducts/Promo6.jpg"),
tk.getImage("PromotionProducts/Promo7.jpg"),
tk.getImage("PromotionProducts/Promo1.jpg"),
tk.getImage("PromotionProducts/Promo2.jpg"),
tk.getImage("PromotionProducts/Promo3.jpg"),
tk.getImage("PromotionProducts/Promo4.jpg"),
tk.getImage("PromotionProducts/Promo5.jpg"),
tk.getImage("PromotionProducts/Promo6.jpg"),
};

```

```

public TableOne(String s, TableFiveSix drawArea){
    //setLayout(new BorderLayout());
    setBackground(SystemColor.activeCaptionBorder);
    buttonPanel = new JPanel();
    buttonBorder = BorderFactory.createLineBorder(Color.white);
    icon = new ImageIcon("Images/"+s);
    button = new JButton(icon);
    buttonPanel.add(button);
    button.setBorder(buttonBorder);
    add(buttonPanel,FlowLayout.LEFT);
    button.addActionListener(new ShowProduct());
    picArea = drawArea;
    addMouseMotionListener(this);
}

class ShowProduct implements ActionListener{
    public void actionPerformed(ActionEvent e){
        picArea.setOtherImage(str);
    }
}

public void mouseMoved(MouseEvent evt){
    int x = evt.getX();
    int y = evt.getY();
    in = inImage(x,y);
    if(in == true)

```



```

        setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    else
        setCursor(Cursor.getDefaultCursor());
}

public void mouseDragged(MouseEvent evt) {
public boolean inImage(int x,int y){
    Point coordinates = buttonPanel.getLocation();
    int xOrdinate = coordinates.x;
    int yOrdinate = coordinates.y;
    Dimension size = button.getSize();
    int buttonWidth = size.width;
    int buttonHeight = size.height;
    int boundWidth = xOrdinate + buttonWidth;
    int boundHeight = yOrdinate + buttonHeight;
    if((x<=boundWidth)&&(y<=boundHeight))
        return true;
    else
        return false;
    }
}
} //end of class Pictures

```

1.3 TableThree.java

```

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;

//class for vertical 5 pictures
class TableThree extends JPanel{
    Toolkit tk = Toolkit.getDefaultToolkit();
    Icon [] images ={
        new ImageIcon("Images/"+"Cs_shelf_1.gif"),
        new ImageIcon("Images/"+"Engineering_Shelf_1.gif"),
        new ImageIcon("Images/"+"Medicine_Shelf_1.gif"),
        new ImageIcon("Images/"+"Chemistry_shelf_1.gif"),
        new ImageIcon("Images/"+"Geology_Shelf_1.gif"),
        new ImageIcon("Images/"+"History_Shelf_1.gif"),
        new ImageIcon("Images/Philosophy_Shelf_1.gif"),
    };
}

```

```

Icon [] tabImages ={
    new ImageIcon("Images/"+"Cs_Tab_2.gif"),
    new ImageIcon("Images/"+"Engineering_Tab_2.gif"),
    new ImageIcon("Images/"+"Medicine_Tab_1.gif"),
    new ImageIcon("Images/"+"Chemistry_Tab_1.gif"),
    new ImageIcon("Images/"+"Geology_Tab_1.gif"),
    new ImageIcon("Images/"+"History_Tab_1.gif"),
    new ImageIcon("Images/Philosophy_Tab_1.gif"),

};

JButton [] buttons = new JButton[images.length];
Image [] compImages ={

    tk.getImage("Images/"+"OOP_Shelf_1.jpg"),
    tk.getImage("Images/"+"AI_Shelf_1.jpg"),
    tk.getImage("Images/"+"HCI_Shelf_1.jpg"),
    tk.getImage("Images/"+"User_Sheif.jpg"),

};

Image [] engineeringImages ={
    tk.getImage("Images/"+"User_Shelf.jpg"),
    tk.getImage("Images/"+"User_Shelf.jpg"),
    tk.getImage("Images/"+"User_Shelf.jpg"),
    tk.getImage("Images/"+"User_Shelf.jpg"),

};

Image [] mathImages ={tk.getImage("Images/welcomeb.gif"),};
Image [] mathProducts ={

    tk.getImage("Images/officebook.gif"),

};

Image [] barImages = {
    tk.getImage("Images/VerComputerBar.gif"),
    tk.getImage("Images/HorComputerBar.gif"),
    tk.getImage("Images/VerGreenBar.gif"),//engineering
    tk.getImage("Images/HorGreenBar.gif"),
    tk.getImage("Images/VerPinkBar.gif"),//medicine
    tk.getImage("Images/HorPinkBar.gif"),
    tk.getImage("Images/VerChemistryBar.gif"),
    tk.getImage("Images/HorChemistryBar.gif"),
    tk.getImage("Images/VerGeologyBar.gif"),
    tk.getImage("Images/HorGeologyBar.gif"),
    tk.getImage("Images/VerHistoryBar.gif"),

```

```

        tk.getImage("Images/HorHistoryBar.gif"),
        tk.getImage("Images/VerPhilosophyBar.gif"),
        tk.getImage("Images/HorPhilosophyBar.gif"),
    };

    TableFour showShelves = null;
    BarOne bar1 = null;
    BarTwo bar2 = null;
    JPanel holdPanel = new JPanel(new GridLayout(images.length,1,0,0));
    Border buttonBorder = BorderFactory.createEmptyBorder();
    TableFiveSix product;
    public TableThree(TableFour subshelves,BarOne barOne,
        BarTwo barTwo,TableFiveSix pro){
        holdPanel.setBackground(SystemColor.activeCaptionBorder);
        for(int i =0;i<images.length;++i){
            buttons[i] = new JButton(images[i]);
        }
        showShelves = subshelves;
        bar1 = barOne;
        bar2 = barTwo;
        for(int i=0; i<images.length; ++i){
            holdPanel.add(buttons[i]);
            buttons[i].setVerticalAlignment(JButton.TOP);
            buttons[i].setHorizontalAlignment(JButton.LEFT);
            buttons[i].addActionListener(new showShelf());
            buttons[i].setBorder(buttonBorder);
            buttons[i].setPressedIcon(tabImages[i]);
            add(holdPanel);
        }
        product = pro;
    }

    class showShelf implements ActionListener{

        public void actionPerformed(ActionEvent e){
            if(e.getSource()== buttons[0]){
                showShelves.setImagesIcon(compImages);
                bar1.setNewImage(barImages[0]);
                bar2.setNewImage(barImages[1]);
            }
            else if(e.getSource() == buttons[1]){
                showShelves.setImagesIcon(engineeringImages);
                bar1.setNewImage(barImages[2]);
                bar2.setNewImage(barImages[3]);
            }
            else if(e.getSource() == buttons[2]){

```

```

        showShelves.setImagesIcon(mathImages);
        product.setOtherImage(mathProducts);
        bar1.setNewImage(barImages[4]);
        bar2.setNewImage(barImages[5]);
    }
    else if(e.getSource() == buttons[3]){
        showShelves.setImagesIcon(engineeringImages);
        bar1.setNewImage(barImages[6]);
        bar2.setNewImage(barImages[7]);
    }
    else if(e.getSource() == buttons[4]){
        showShelves.setImagesIcon(compImages);
        bar1.setNewImage(barImages[8]);
        bar2.setNewImage(barImages[9]);
    }
    else if(e.getSource() == buttons[5]){
        showShelves.setImagesIcon(engineeringImages);
        bar1.setNewImage(barImages[10]);
        bar2.setNewImage(barImages[11]);
    }
    else if(e.getSource() == buttons[6]){
        showShelves.setImagesIcon(engineeringImages);
        bar1.setNewImage(barImages[12]);
        bar2.setNewImage(barImages[13]);
    }
    }
}
} //end of class

```

1.4 TableFour.java

```

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;
import java.io.*;

public class TableFour extends JPanel implements MouseMotionListener{
    Toolkit tk = Toolkit.getDefaultToolkit();
    Image [] images;
    Image leftArrow = tk.getImage("Images/Arrow_Left.gif");
    Image rightArrow = tk.getImage("Images/Right_Arrow2.gif");
    Point [] position;
    TableFiveSix product;
    Image [] oop = {

```

```

        tk.getImage("CsOOPproducts/oop1.gif"),
        tk.getImage("CsOOPproducts/oop2.gif"),
        tk.getImage("CsOOPproducts/oop3.gif"),
        tk.getImage("CsOOPproducts/oop4.gif"),
        tk.getImage("CsOOPproducts/oop5.gif"),
        tk.getImage("CsOOPproducts/oop6.gif"),
        tk.getImage("CsOOPproducts/oop7.gif"),
        tk.getImage("CsOOPproducts/oop1.gif"),
        tk.getImage("CsOOPproducts/oop2.gif"),
        tk.getImage("CsOOPproducts/oop3.gif"),
        tk.getImage("CsOOPproducts/oop4.gif"),
        tk.getImage("CsOOPproducts/oop5.gif"),
        tk.getImage("CsOOPproducts/oop6.gif"),
        tk.getImage("CsOOPproducts/oop7.gif"),

};

public TableFour(Image [] shelves,TableFiveSix showProduct){
    this.setPreferredSize(new Dimension(360,100));
    images = shelves;
    product = showProduct;
    position = new Point[images.length];
    addMouseListener(new MouseAdapter()
        {
            int in = 0;
            int press = 0;
            public void mousePressed(MouseEvent evt)
            {
                int x = evt.getX();
                int y = evt.getY();
                in = findImage(x,y);
                if(in == 0)
                    product.setOtherImage(oop);
            }
        });

    addMouseMotionListener(this);
}

public void paintComponent(Graphics g){
    super.paintComponent(g);
    int xPosition = this.getLocation().x+10;
    int yPosition = this.getLocation().y;
    for (int i=0;i<images.length;++i){

```

```

        g.drawImage(images[i],xPosition,yPosition,this);
        position[i] = new Point(xPosition,yPosition);
        xPosition += 100;
    }
}

public void mouseMoved(MouseEvent evt)
{
    int x = evt.getX();
    int y = evt.getY();
    if(findImage(x,y)>=0)

        setCursor(Cursor.getPredefinedCursor
                    (Cursor.HAND_CURSOR));
    else
        setCursor(Cursor.getDefaultCursor());

}

public void mouseDragged(MouseEvent evt){ }
public void setImagesIcon(Image [] otherImages){
    images = otherImages;
    repaint();
}

public int findImage(int x, int y){
    for(int i=0;i<images.length;++i){
        if((position[i].x <= x)&&(x <=
            position[i].x+images[i].getWidth(this))
            &&(position[i].y <= y)&&(y <=
            position[i].y+images[i].getHeight(this)))
            return i;
        }
    return -1;
}
}
} //end of class

```

1.5 ToolBar.java

```
// RealThing based ToolBar.java
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.border.*;

public class ToolBar extends JPanel{
    ImageIcon [] pictures = {
        new ImageIcon("Images/deliv.gif"),
        new ImageIcon("Images/Payment.gif"),
        new ImageIcon("Images/master_card.gif"),
        new ImageIcon("Images/newInvoice.jpg"),
        new ImageIcon("Images/cash.gif"),
        new ImageIcon("Images/Bag.gif"),
        new ImageIcon("Images/product_View.jpg"),
        new ImageIcon("Images/adviser.gif"),
        new ImageIcon("Images/userPref.gif"),
        new ImageIcon("Images/Tool_Razibul.gif"),
    };

    String [] names = {"Delivery", "Bill", "Payment", "Invoice", "Cash", "Bag",
        "Viewer", "Advisor", "Shopping Record", "User"};
    JLabel [] labels = new JLabel[names.length];
    JButton [] buttons = new JButton[pictures.length];
    Border buttonBorder = BorderFactory.createEmptyBorder();
    Box deliveryLabel = Box.createVerticalBox();
    Box billLabel = Box.createVerticalBox();
    Box paymentLabel = Box.createVerticalBox();
    Box invoiceLabel = Box.createVerticalBox();
    Box cashLabel = Box.createVerticalBox();
    Box bagLabel = Box.createVerticalBox();
    Box viewLabel = Box.createVerticalBox();
    Box advisorLabel = Box.createVerticalBox();
    Box recordLabel = Box.createVerticalBox();
    Box userLabel = Box.createVerticalBox();

    public ToolBar(){
        setBackground(SystemColor.activeCaptionBorder);
        for(int i=0;i<pictures.length;++i){
            buttons[i] = new JButton(pictures[i]);
            buttons[i].setBorder(buttonBorder);

            buttons[i].setBackground(SystemColor.activeCaptionBorder);
        }
    }
}
```

```

        buttons[i].addActionListener(new Purchase());
        labels[i] = new JLabel(names[i],SwingConstants.RIGHT);
        switch(i){
        case 0: addButtonLabel(deliveryLabel,buttons[i],labels[i]);break;
        case 1: addButtonLabel(billLabel,buttons[i],labels[i]);break;
        case 2: addButtonLabel(paymentLabel,buttons[i],labels[i]);break;
        case 3: addButtonLabel(invoiceLabel,buttons[i],labels[i]);break;
        case 4: addButtonLabel(cashLabel,buttons[i],labels[i]);break;
        case 5: addButtonLabel(bagLabel,buttons[i],labels[i]);break;
        case 6: addButtonLabel(viewLabel,buttons[i],labels[i]);break;
        case 7: addButtonLabel(advisorLabel,buttons[i],labels[i]);break;
        case 8: addButtonLabel(recordLabel,buttons[i],labels[i]);break;
        case 9: addButtonLabel(userLabel,buttons[i],labels[i]);break;
        }
    }
}

void addButtonLabel(Box box,JButton button,JLabel label){
    box.add(button);
    label.setVerticalTextPosition(JLabel.BOTTOM);
    label.setHorizontalTextPosition(JLabel.CENTER);
    box.add(label);
    add(box);
}

public class Purchase implements ActionListener{
    public void actionPerformed(ActionEvent e){
        if (e.getSource() == buttons[pictures.length-1]){
            UserDataFrame user = new UserDataFrame();
            user.show();
        }
        else if(e.getSource() == buttons[0]){
            DeliveryFrame delivery = new DeliveryFrame();
            delivery.show();
        }
        else if(e.getSource() == buttons[1]){
            BillFrame bill = new BillFrame();
            bill.show();
        }
        else if(e.getSource() == buttons[2]){
            PaymentFrame payment = new PaymentFrame();
            payment.show();
        }
        else if(e.getSource() == buttons[3]){
            InvoiceFrame invoice = new InvoiceFrame();
            invoice.show();
        }
    }
}

```



```

        else if(e.getSource() == buttons[4]){
            CashFrame cash = new CashFrame();
            cash.show();
        }
        else if(e.getSource() == buttons[5]){
            IconBagListTable bagFrame = new IconBagListTable();
        }
        else if(e.getSource() == buttons[6]){
            ProductViewerFrame viewer = new ProductViewerFrame();
            viewer.show();
        }
        else if(e.getSource() == buttons[7]){
            AdvisorFrame advisor = new AdvisorFrame();
            advisor.show();
        }
        else if(e.getSource() == buttons[8]){
            ShoppingRecordFrame record = new ShoppingRecordFrame();
            record.show();
        }
    }
}

public static void main(String [] args){
    JFrame frame = new JFrame();
   ToolBar tbt = new ButtonTableTwo();
    frame.getContentPane().add(tbt);
    frame.show();
}
} //end of class

```

1.6 ImageIcon.java

```

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import java.io.*;
import javax.swing.border.*;

public class ImageIcon extends JPanel{
    Toolkit tk = Toolkit.getDefaultToolkit();
    JButton text;
    JButton changeButton;
    JButton previous;
    JButton next;
}

```

```

JPanel buttonPanel;
int maxImages = 100;
Image [] imas;
int [] imageWidth;
int [] imageHeight;
boolean change = false;
boolean isText = false;
int totalPages = 0;
int currentPage = 1;
int beginBook = 1;
final int numOfDispOnce = 8; //8 books will be displayed once
JFrame dialogFrame;
static Icon rightArrow = new ImageIcon("Images/Forward.gif");
static Icon leftArrow = new ImageIcon("Images/Backward.gif");

public ImageIcon(Image [] images,JFrame frame){
    setBackground(SystemColor.activeCaptionBorder);
    this.setPreferredSize(new Dimension(360,200));
    setLayout(new BorderLayout());
    buttonPanel = new JPanel();
    buttonPanel.setBackground(SystemColor.activeCaptionBorder);
    text = new JButton("Text/Image");
    text.setBackground(SystemColor.activeCaptionBorder);
    text.addActionListener(new ChangeToText());

    changeButton = new JButton("Icon/Image");
    changeButton.setBackground(SystemColor.activeCaptionBorder);
    changeButton.addActionListener(new ChangeImageIcon());
}

```

```

previous = new JButton("Back",leftArrow);
previous.setBackground(SystemColor.activeCaptionBorder);
previous.addActionListener(new ShowPreItems());
next = new JButton("Forward",rightArrow);
next.setBackground(SystemColor.activeCaptionBorder);
next.addActionListener(new ShowMoreItems());
buttonPanel.add(text,FlowLayout.LEFT);
buttonPanel.add(changeButton);
buttonPanel.add(previous);
buttonPanel.add(next);
add(buttonPanel,"North");
imas = images;
dialogFrame = frame;
imageWidth = new int[imas.length];
imageHeight = new int[imas.length];
if(imas.length%numOfDispOnce == 0)
    totalPages = imas.length/numOfDispOnce;
else
    totalPages = imas.length/numOfDispOnce+1;
}

public void setOtherImage(Image [] anotherIma){

    imas = anotherIma;
    repaint();

}

public void paintComponent(Graphics g){

    super.paintComponent(g);
    //int xPosition = text.getLocation().x-50;//50
    int xPosition = this.getLocation().x+20;
    int yPosition = 50;
if(isText == false){
    if(change == false){
        for (int i=0;i<imas.length;++i){
            imageWidth[i] = imas[i].getWidth(this);
            imageHeight[i] = imas[i].getHeight(this);
        }
    }
}
}

```

```

else{
    for (int i=0;i<imas.length;++i){
        imageWidth[i] = imas[i].getWidth(this)/2;
        imageHeight[i] = imas[i].getHeight(this)/2;
    }
}

for(beginBook = (currentPage-1)*8+1;
((beginBook <= imas.length)&&(beginBook
<=currentPage*8));++beginBook){
    if((((currentPage-1)*8+1) <= beginBook)&&
( beginBook <= ((currentPage-1)*8+4))){

        g.drawImage(imas
[beginBook-1],
xPosition,yPosition,imageWidth
[beginBook-1],imageHeight[beginBook-1],this);
        xPosition += 110;
    }
    if((((currentPage-1)*8+5) == beginBook){
        //xPosition = text.getLocation().x-50;
        xPosition = this.getLocation().x+20;
        yPosition = 200;
        g.drawImage(imas[beginBook-1],
xPosition,yPosition,imageWidth
[beginBook-1],imageHeight[beginBook-1],this);
    }
    if((((currentPage-1)*8+5) < beginBook)&&
( beginBook <= ((currentPage-1)*8+8))){

        xPosition += 110;
        g.drawImage(imas[beginBook-1],
xPosition,yPosition,imageWidth
[beginBook-1],imageHeight[beginBook-1],this);
    }

}

} //end of for
} //end of if
else
g.drawString("Core Java 2,Volume 1:
Fundamentals...",xPosition,yPosition);

} //end of paintComponent()

```

```

class ChangeToText implements ActionListener{
    public void actionPerformed(ActionEvent e){

        if(isText == false){
            isText = true;
            text.setLabel("Images");
        }
        else{
            isText = false;
            text.setLabel("Text");
        }
        repaint();
    }
}

```

```

class ChangeImageIcon implements ActionListener{
    public void actionPerformed(ActionEvent e){
        if(change == false){
            change = true;
            changeButton.setLabel("Image");
        }
        else{
            change = false;
            changeButton.setLabel("Icon");
        }
        repaint();
    }
}

```

}//end of listener class

```

class ShowMoreItems implements ActionListener{
    public void actionPerformed(ActionEvent e){
        if(currentPage < totalPages){
            ++currentPage;
            repaint();
        }

        else{
            FinishDialog fini = new FinishDialog(dialogFrame);
            fini.show();
        }//end of else
    }//end of function
}//end of class

```

```

class FinishDialog extends JDialog{
    public FinishDialog(JFrame parent){
        super(parent,"Finish",true);
        JLabel finish = new JLabel("The End",JLabel.CENTER);
        getContentPane().add(finish,"Center");
        JPanel okPanel = new JPanel();
        JButton ok = new JButton("Ok");
        okPanel.add(ok);
        getContentPane().add(okPanel,"South");
        ok.addActionListener(new ActionListener()
        {public void actionPerformed(ActionEvent e)
            {setVisible(false);}
        });
        setSize(250,150);
    }//end of constructor

} //end of FinishDialog class

class ShowPreItems implements ActionListener{
    public void actionPerformed(ActionEvent e){
        if(currentPage != 1){
            --currentPage;
            repaint();
        }
        else{
            FinishDialog fini = new FinishDialog(dialogFrame);
            fini.show();
        } //end of else
    } //end of function
} //end of class
} //end of class

```

1.7 UserShelfListTable.java

```

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.*;

public class UserShelfListTable {
    public void userShelf(){
        JFrame f = new JFrame("User Shelf Table");
        JTable tbl = new JTable(new UserShelfTableModel());
        tbl.setDefaultRenderer(java.lang.Number.class,
            new FractionCellRenderer(20, 3, SwingConstants.RIGHT));
    }
}

```

```

        TableColumnModel tcm = tbl.getColumnModel();
        tcm.getColumnModel().setPreferredWidth(150);
        tcm.getColumnModel().setMinWidth(150);
        TextWithIconCellRenderer renderer = new TextWithIconCellRenderer();
        tcm.getColumnModel().setCellRenderer(renderer);

        tbl.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        tbl.setPreferredScrollableViewportSize(tbl.getPreferredSize());

        JScrollPane sp = new JScrollPane(tbl);
        f.getContentPane().add(sp, "Center");
        f.pack();
        f.setVisible(true);
    }
    public UserShelfListTable(){
        userShelf();
    }
}

```

1.8 BagList.java

```

import javax.swing.*;
import javax.swing.table.AbstractTableModel;

public class BagList extends AbstractTableModel {
    protected String[] columnNames =
        {"ProductName", "Quantity", "Price", "Total" };

    public BagList() {
        for (int i = 0; i < data.length; i++) {
            data[i][DIFF_COLUMN] =
                new Double(((Double)data[i][NEW_RATE_COLUMN]).
                    doubleValue() * ((Double)data[i]
                    [OLD_RATE_COLUMN]).doubleValue());
        }
    }

    public int getRowCount() {
        return data.length;
    }

    public int getColumnCount() {
        return COLUMN_COUNT;
    }
}

```

```

    }

    public Object getValueAt(int row, int column) {
        return data[row][column];
    }

    public Class getColumnClass(int column) {
        return (data[0][column]).getClass();
    }

    public String getColumnName(int column) {
        return columnNames[column];
    }

    protected static final int OLD_RATE_COLUMN = 1;
    protected static final int NEW_RATE_COLUMN = 2;
    protected static final int DIFF_COLUMN = 3;
    protected static final int COLUMN_COUNT = 4;
    protected static final Class thisClass = BagList.class;
    protected Object[][] data = new Object[][] {
        { new DataWithIcon("Java builder Guide",
            new ImageIcon(thisClass.getResource("images/Promo11.gif")),
            new Double(1), new Double(37.65), null },
        { new DataWithIcon("VRML",
            new ImageIcon(thisClass.getResource("images/Promo22.gif ")),
            new Double(2), new Double(61.04), null },
        { new DataWithIcon("Java Developers",
            new ImageIcon(thisClass.getResource("images/Promo33.gif")),
            new Double(4), new Double(50.11), null },
        { new DataWithIcon("Gold Fusion",
            new ImageIcon(thisClass.getResource("images/Promo44.gif ")),
            new Double(6), new Double(100.1) , null },
        { new DataWithIcon("Tango",
            new ImageIcon(thisClass.getResource("images/Promo55.gif ")),
            new Double(8), new Double(82.1), null },
        { new DataWithIcon("JBulider",
            new ImageIcon(thisClass.getResource("images/Promo66.gif ")),
            new Double(1), new Double(82.23), null },
        { new DataWithIcon("Power Bulider",
            new ImageIcon(thisClass.getResource("images/Promo77.gif")),
            new Double(4), new Double(21.4), null }
    };
}

```


1.9 InvoiceTable.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.border.*;

class InvoiceTable extends AbstractTableModel{
    private String [] columnNames = {"Product Name","ISBN","
        Unit Price","Quantity","Cumulative total","TPS(7%)","
        TVQ(7.5%)","Grand Total"};
    Object [][] data ={
        {"Core Java","0130819336",new Double(30.09),new Integer(1),
        new Double(30.09),new Double(2.11),new Double(2.26),
        new Double(34.46)},
        {"Core Java","0130819336",new Double(30.09),new Integer(1),
        new Double(30.09),new Double(2.11),new Double(2.26),
        new Double(34.46)},
        {"Core Java","0130819336",new Double(30.09),new Integer(1),
        new Double(30.09),new Double(2.11),new Double(2.26),
        new Double(34.46)},
        {"Core Java","0130819336",new Double(30.09),new Integer(1),
        new Double(30.09),new Double(2.11),new Double(2.26),
        new Double(34.46)},
        };

    public int getRowCount(){
        return data.length;
    }

    public int getColumnCount(){
        return columnNames.length;
    }
    public String getColumnName(int c){

        return columnNames[c];
    }

    public boolean isCellEditable(int row, int col) {
        return false;
    }
    public Object getValueAt(int r,int c){
        return data[r][c];
    }
}
```

```

        public void setValueAt(Object val,int row,int col){
            data[row][col] = val;
            fireTableCellUpdated(row, col);
        }
    }

    public class InvoiceFrame extends JFrame implements ActionListener{
        Box winBox = Box.createVerticalBox();
        Box infoBox = Box.createHorizontalBox();
        JPanel buttonPanel;
        JButton ok;
        JButton cancel;

        JLabel [] labels = {
            new JLabel("Customer Name",SwingConstants.RIGHT),
            new JLabel("Customer Address",SwingConstants.RIGHT),
        };

        JTextField nameField;
        JTextField addressField;
        JPanel labelPanel;
        JPanel textPanel;
        JPanel [] textPanels = new JPanel[labels.length];

        Border topBorder = BorderFactory.createLineBorder(Color.white);
        JLabel id;
        JTextField idField;
        JButton goButton;

        public InvoiceFrame(){
            setTitle("Invoice Information");
            setSize(400,300);

            buttonPanel = new JPanel();
            ok = new JButton("Ok");
            cancel = new JButton("Cancel");

            labelPanel = new JPanel(new GridLayout(labels.length,1,0,0));
            textPanel = new JPanel(new GridLayout(labels.length,1,0,0));

            nameField = new JTextField(20);
            addressField = new JTextField(20);

            JPanel firstPanel = new JPanel();
            Border buttonBorder = BorderFactory.createEmptyBorder();

```

```

id = new JLabel("Customer ID");
idField = new JTextField(8);
Icon go = new ImageIcon("Images/rightarrow.gif");
goButton = new JButton(go);
//goButton.addActionListener(new FindCustomer());
goButton.setBorder(buttonBorder);
firstPanel.add(id,FlowLayout.LEFT);
firstPanel.add(idField);
firstPanel.add(goButton);
firstPanel.setBorder(topBorder);
getContentPane().add(firstPanel,"North");

for(int i=0;i<labels.length;++i)
    labelPanel.add(labels[i]);

for(int i=0;i<labels.length;++i)
    textPanels[i] = new JPanel();

    textPanels[0].add(nameField);
    textPanel.add(textPanels[0]);

    textPanels[1].add(addressField);
    textPanel.add(textPanels[1]);

    infoBox.add(labelPanel);
    infoBox.add(textPanel);
    winBox.add(infoBox);

    InvoiceTable invoiceTableModel = new InvoiceTable();
    JTable invoiceTable = new JTable(invoiceTableModel);
    invoiceTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    invoiceTable.setPreferredScrollableViewportSize
    (invoiceTable.getPreferredSize());
    winBox.add(new JScrollPane(invoiceTable),"Center");
    getContentPane().add(winBox,"Center");

    buttonPanel.add(ok);
    buttonPanel.add(cancel);
    ok.addActionListener(this);
    cancel.addActionListener(this);
    getContentPane().add(buttonPanel,"South");
}
public void actionPerformed(ActionEvent e){
setVisible(false);
}

```

```

public static void main(String [] args){
    InvoiceFrame r = new InvoiceFrame();
    r.show();
}

```

}//end of class

1.10 UserInfo.java

```

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;

```

```

class UserInfo{
    private Name nm;
    private Address ad;
    private Telephone phone;
    private int customerId;
    private char [] pwd;

```

```

//this array is tempory.it is used to anoly a DB for the password check
int [] temp = { 1,2,3,4,5,6,7,8,9};

```

```

//constructors

```

```

public UserInfo(Name na,Address addr,Telephone tel,int id,char [] pd){
    nm = na;
    ad = addr;
    phone = tel;
    customerId = id;
    pwd = pd;
}

```

```

public UserInfo(Name na,Address addr,Telephone tel,char [] pd){
    nm = na;
    ad = addr;
    phone = tel;
    pwd = pd;
}

```

```

public Name getName(){return nm;}
public Address getAddress(){return ad;}
public Telephone getPhone(){return phone;}
public int getCustomerId(){return customerId;}

```

```
}//
```

1.11 UserDataFrame.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.event.*;
import java.io.*;

public class UserDataFrame extends JFrame
    implements ActionListener{
    private JFrame userData;
    private Box userInforBox;

    JPanel leftPanel;
    JPanel rightPanel;
    private JLabel lastName;
    private JLabel firstName;
    private JLabel midName;
    private JLabel street;
    private JLabel cityState;
    private JLabel country;
    private JLabel postCode;
    private JLabel areaCode;
    private JLabel number;
    private JButton ok;
    private JButton cancel;
    private JLabel id;
    private JLabel sin;
    private JLabel newSin;

    private JPasswordField password;
    private JPasswordField secondPassword;
    private UserInfo data;
    Name tempName;
    Address tempAddr;
    Telephone tempTel;
    //Telephone alterTel;
    char [] tempPassword;
    char [] rePassword;
    JTextField [] field;
    JPanel [] fieldPanel;
```

```

public UserDataFrame() {
    setTitle("UserInformation");
    setResizable(true);
    setSize(400,500);
    setFont(new Font("SansSerif",Font.BOLD,14));
    userInforBox = Box.createHorizontalBox();
    JLabel [] labels = {
        new JLabel("First Name",SwingConstants.RIGHT),
        new JLabel("Middle Name",SwingConstants.RIGHT),
        new JLabel("Last Name",SwingConstants.RIGHT),
        new JLabel("Street",SwingConstants.RIGHT),
        new JLabel("City/State/Province",SwingConstants.RIGHT),
        new JLabel("Country",SwingConstants.RIGHT),
        new JLabel("Post Code",SwingConstants.RIGHT),
        new JLabel("Area Code",SwingConstants.RIGHT),
        new JLabel("Telephone Number",SwingConstants.RIGHT),
        //new JLabel("Alternate Telephone",SwingConstants.RIGHT),
        new JLabel("Password",SwingConstants.RIGHT),
        new JLabel("Reenter Password",SwingConstants.RIGHT),
    };

    leftPanel = new JPanel(new GridLayout(labels.length,1,0,0));
    rightPanel = new JPanel(new GridLayout(labels.length,1,0,0));
    for(int i=0;i<labels.length;++i)
        leftPanel.add(labels[i]);

    fieldPanel = new JPanel[labels.length];
    for(int i=0;i<labels.length;++i)
        fieldPanel[i] = new JPanel();
    field = new JTextField[labels.length];
    for(int i=0;i<labels.length-2;++i)
        field[i] = new JTextField(15);

    password = new JPasswordField(15);
    secondPassword = new JPasswordField(15);
    int i;
    for(i=0;i<labels.length-2;++i){

        fieldPanel[i].add(field[i],FlowLayout.LEFT);
        rightPanel.add(fieldPanel[i]);

    }

    fieldPanel[i].add(password,FlowLayout.LEFT);
    rightPanel.add(fieldPanel[i]);

```

```

        fieldPanel[++i].add(secondPassword,FlowLayout.LEFT);
        rightPanel.add(fieldPanel[i]);
        JPanel buttonPanel = new JPanel();
        ok = new JButton("Ok");
        buttonPanel.add(ok);
        cancel = new JButton("Cancel");
        buttonPanel.add(cancel);
        ok.addActionListener(this);
        cancel.addActionListener(this);
        userInfoBox.add(leftPanel);
        userInfoBox.add(rightPanel);
        Box winBox = Box.createVerticalBox();
        winBox.add(userInfoBox);
        winBox.add(buttonPanel);
        getContentPane().add(winBox);
    } //end of constructor

    public static void main(String [] args){
        UserDataFrame user = new UserDataFrame();
        user.show();
    }

    public void initialize(){
        tempName = new Name(field[0].getText().trim(),
            field[1].getText().trim(),field[2].getText().trim());
        tempAddr = new Address(field[3].getText().trim(),
            field[4].getText().trim(),field[5].getText().trim(),field[6].getText().trim());
        int area = Integer.parseInt(field[7].getText().trim());
        int telNum = Integer.parseInt(field[8].getText().trim());
        tempTel = new Telephone(area,telNum);
        //tempCustomerId = Integer.parseInt(field[9].getText().trim());
        tempPassword = password.getPassword();
        rePassword = secondPassword.getPassword();
        int i;
        for(i=0;i<tempPassword.length;++i){

            if(tempPassword[i] == rePassword[i])
                continue;
            else
                break;
        }
        if(i != tempPassword.length){
            MessageDialog wrongMessage = new MessageDialog(this);
            wrongMessage.show();
        }
        else{

```

```

        OutputStream out;

        try{
            out = new FileOutputStream("userdata.dat",true);
            out.write(tempName.getName().getBytes());
            out.write(";".getBytes());
            out.write(tempAddr.getAddress().getBytes());
            out.write(";".getBytes());
            out.write(tempTel.getTel().getBytes());
            out.write(("#+\r").getBytes());
            out.close();

        }catch(IOException e){ }
    }

    public void actionPerformed(ActionEvent e){
        if(e.getSource() == ok){
            initialize();
            data = new UserInfo(tempName,tempAddr,tempTel,rePassword);
            setVisible(false);

        }
        else if(e.getSource() == cancel)

            setVisible(false);

    } //end of actionPerformed()
}

class MessageDialog extends JDialog{
    public MessageDialog(JFrame parent){
        super(parent,"Finish",true);
        JLabel finish = new JLabel("Wrong
            Password",JLabel.CENTER);
        getContentPane().add(finish,"Center");
        JPanel okPanel = new JPanel();
        JButton ok = new JButton("Ok");
        okPanel.add(ok);
        getContentPane().add(okPanel,"South");
        ok.addActionListener(new ActionListener()
        {public void actionPerformed(ActionEvent e)
            {
                setVisible(false);
            }
        }
    }
}

```



```

    });
    setSize(250,150);
} //end of constructor

```

```

}

```

1.12 ShoppingRecord.java

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.io.*;
import javax.swing.border.*;

```

```

class ShoppingRecord{
    private static final int maxEntry = 200;
    private long [] customerId;
    private Date [] buyDate = new Date[maxEntry];
    private String [] bookName = new String[maxEntry];
    private float [] amount = new float[maxEntry];
    private int [] quantity = new int[maxEntry];
    private String [] payMethod = new String[maxEntry];
    private String [] status = new
        String[maxEntry]; //delivering,processing,unpaid,etc.
    private int entryNum = 0; //should be 0,to show the table,
        //assume already has 10 rows
    private int numOfAccess = 0;//keeps tracks of the number of accesses of one user

    public ShoppingRecord(int id,Date d,String nm,float money,
        int q,String how,String s){

        customerId[entryNum++] = id;
        buyDate[entryNum++] = d;
        bookName[entryNum++] = nm;
        quantity[entryNum++] = q;
        amount[entryNum++] = money;
        payMethod[entryNum++] = how;
        status[entryNum++] = s;

    }
    public long getId(int i){return customerId[i];}
    public Date getDate(int i){return buyDate[i];}
    public String getBookName(int i){return bookName[i];}
}

```

```

    public int getQuantity(int i){return quantity[i];}
    public float getAmount(int i){return amount[i];}
    public String getPayMethod(int i){return payMethod[i];}
    public String getStatus(int i){return status[i];}
    public int getAccessNumber(){return numOfAccess;}
    public void incrementNumOfAccess(){++numOfAccess;}
    public int getEntryNumber(){return entryNum;}
}

```

```

class RecordTable extends AbstractTableModel{
    private String [] columnNames = {"Customer ID", "Buy Date", "
Books", "Quantity", "Total Amount", "Payment", "Status"};
    Object [][] data ={
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
        {"12345", "12 June,2000", "Core Java", "1", "$30.09", "Visa", "on delivery"},
    };

    public int getRowCount(){
        return data.length;
    }

    public int getColumnCount(){
        return columnNames.length;
    }
    public String getColumnName(int c){

        return columnNames[c];

    }
    public boolean isCellEditable(int row, int col) {
        return false;
    }
    public Object getValueAt(int r,int c){
        return data[r][c];
    }
    public void setValueAt(Object val,int row,int col){
        data[row][col] = val;
    }
}

```

```

        fireTableCellUpdated(row, col);
    }
}

public class ShoppingRecordFrame extends JFrame{
    JPanel idPanel;
    JLabel idLabel;
    JTextField idField;
    JButton goButton;
    RecordTable recordTableModel;
    JTable recordTable;

    public ShoppingRecordFrame(){
        setTitle("Shopping Record");
        setSize(500,300);

        idPanel = new JPanel();
        Border buttonBorder = BorderFactory.createEmptyBorder();
        idLabel = new JLabel("Customer ID");
        idField = new JTextField("Input ID",10);
        Icon go = new ImageIcon("Images/rightarrow.gif");
        goButton = new JButton(go);
        goButton.addActionListener(new FindRecord());
        goButton.setBorder(buttonBorder);
        idPanel.add(idLabel,FlowLayout.LEFT);
        idPanel.add(idField);
        idPanel.add(goButton);
        getContentPane().add(idPanel,"North");

        recordTableModel = new RecordTable();
        recordTable = new JTable(recordTableModel);
        recordTable.setPreferredSize
            (new Dimension(500, 50));
        getContentPane().add(new JScrollPane(recordTable),"Center");

    } //end of constructor

    public void clearAll(){
        for(int i =0;i<recordTable.getRowCount();++i){
            for(int j=0;j<recordTable.getColumnCount();++j)
                recordTable.setValueAt(" ",i,j);
        }
    }

    class FindRecord implements ActionListener{

```

```

public void actionPerformed(ActionEvent evt){

    if(evt.getSource() == goButton){
        clearAll();
        InputStream in;
        String buffer;
        String id;
        String date;
        String name;
        String quantity;
        String amount;
        String payment;
        String status;
        int start=0;
        int position=0;
        int col = 0;
        int row = 0;
        try{
            in = new FileInputStream("Record.txt");
            BufferedReader d = new BufferedReader
                (new InputStreamReader(in));
            buffer=d.readLine();
            while(buffer!=null){
                position = buffer.indexOf(";",start);
                id = buffer.substring(start, position);

                String s = idField.getText().trim();
                if(s.equals(id)){
                    recordTable.setValueAt(id,row,col++);
                    start = position+1;
                    position = buffer.indexOf(";",start);
                    date = buffer.substring(start,position);
                    recordTable.setValueAt(date,row,col++);
                    start = position+1;
                    position = buffer.indexOf(";",start);
                    name = buffer.substring(start,position);
                    recordTable.setValueAt(name,row,col++);
                    start = position+1;
                    position = buffer.indexOf(";",start);
                    quantity = buffer.substring(start,position);
                    recordTable.setValueAt(quantity,
                                            row,col++);

                    start = position+1;
                }
            }
        }
    }
}

```

```

        position = buffer.indexOf(";",start);
        amount = buffer.substring(start,position);
        recordTable.setValueAt(amount,row,col++);

        start = position+1;
        position = buffer.indexOf(";",start);
        payment = buffer.substring(start,position);
        recordTable.setValueAt(payment,
                                row,col++);

        start = position+1;
        position = buffer.indexOf("#",start);
        status = buffer.substring(start,position);
        recordTable.setValueAt(status,row,col++);
        ++row;
        col = 0;
    } //end of inner if
    buffer = d.readLine();
    start = 0;
    position = 0;

    } //end of while
    in.close();
} catch(IOException e){ }

    } //end of outer if
} //end of function
}

    public static void main(String [] args){
        ShoppingRecordFrame r = new ShoppingRecordFrame();
        r.show();
    }

} //end of class

```

1.13 Delivery.java

```

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.io.*;

```

```

class Delivery{

    private DeliverList dList;
    private Invoice receipt;

    public Delivery(DeliverList list, Invoice rp){

        dList = list;
        receipt = rp;

    }

    public Invoice getInvoice(){return receipt;}
    public DeliverList getDeliverList(){return dList;}
}

public class DeliveryFrame extends JFrame implements ActionListener{

    Box winBox = Box.createHorizontalBox();

    JPanel buttonPanel = new JPanel();
    JButton ok = new JButton("Ok");
    JButton cancel = new JButton("Cancel");

    JLabel [] labels = {
        new JLabel("Customer Name", SwingConstants.RIGHT),
        new JLabel("Telephone", SwingConstants.RIGHT),
        new JLabel("Alternate
            Telephone", SwingConstants.RIGHT),
        new JLabel("Customer Address", SwingConstants.RIGHT),
        new JLabel("Delivery Date", SwingConstants.RIGHT),
        new JLabel("Delivery Time", SwingConstants.RIGHT),
        new JLabel("Delivery Method", SwingConstants.RIGHT),
        new JLabel("Different Address", SwingConstants.RIGHT),
    };

    JPanel labelPanel = new JPanel(new
        GridLayout(labels.length, 1, 0, 0));
    JPanel textPanel = new JPanel(new
        GridLayout(labels.length, 1, 0, 0));

    JTextField nameField = new JTextField(20);
    JTextField telField = new JTextField(20);
    JTextField alterTelField = new JTextField(20);
    JTextField addressField = new JTextField(20);
    JTextField dayField = new JTextField("Day", 6);
    JTextField minField = new JTextField("Minute", 6);
    JTextField secondAddr = new JTextField("", 20);

    JPanel [] textPanels = new JPanel[labels.length];
    JComboBox month;
    JComboBox year;
}

```

```

JComboBox hour;
JComboBox ampm;
JComboBox shipMethod;
Border topBorder =
    BorderFactory.createLineBorder(Color.white);
JLabel id;
JTextField idField;
JButton goButton;

public DeliveryFrame() {
    setTitle("UserInformation");
    setResizable(true);
    setSize(500,400);
    setFont(new Font("SansSerif",Font.BOLD,14));

JPanel firstPanel = new JPanel();
Border buttonBorder = BorderFactory.createEmptyBorder();
id = new JLabel("Customer ID");
idField = new JTextField(8);
Icon go = new ImageIcon("Images/rightarrow.gif");
goButton = new JButton(go);
goButton.addActionListener(new FindCustomer());
goButton.setBorder(buttonBorder);
firstPanel.add(id,FlowLayout.LEFT);
firstPanel.add(idField);
firstPanel.add(goButton);
firstPanel.setBorder(topBorder);
getContentPane().add(firstPanel,"North");

    for(int i=0;i<labels.length;++i)
        labelPanel.add(labels[i]);

    for(int i=0;i<labels.length;++i)
        textPanels[i] = new JPanel();

month = new JComboBox();
month.addItem("January");
month.addItem("Febuary");
month.addItem("March");
month.addItem("April");
month.addItem("May");
month.addItem("June");
month.addItem("July");
month.addItem("August");
month.addItem("September");
month.addItem("October");
month.addItem("November");
month.addItem("December");

year = new JComboBox();
year.addItem("1999");
year.addItem("2000");
year.addItem("2001");
year.addItem("2002");
year.addItem("2003");

```

```

year.addItem("2004");
year.addItem("2005");
year.setEditable(true);

hour = new JComboBox();
hour.addItem("0");
hour.addItem("1");
hour.addItem("2");
hour.addItem("3");
hour.addItem("4");
hour.addItem("5");
hour.addItem("6");
hour.addItem("7");
hour.addItem("8");
hour.addItem("9");
hour.addItem("10");
hour.addItem("11");
hour.addItem("12");

ampm = new JComboBox();
ampm.addItem("AM");
ampm.addItem("PM");

shipMethod = new JComboBox();
shipMethod.addItem("Pick up at the store");
shipMethod.addItem("Express Post--By fast mail");
shipMethod.addItem("Purolator--Available
                    for delivery 8:00AM-5:PM");

textPanels[0].add(nameField);
textPanel.add(textPanels[0]);

textPanels[1].add(telField);
textPanel.add(textPanels[1]);

textPanels[2].add(alterTelField);
textPanel.add(textPanels[2]);

textPanels[3].add(addressField);
textPanel.add(textPanels[3]);

textPanels[4].add(month,FlowLayout.LEFT);
textPanels[4].add(dayField);
textPanels[4].add(year);
textPanel.add(textPanels[4]);

textPanels[5].add(hour,FlowLayout.LEFT);
textPanels[5].add(minField);
textPanels[5].add(ampm);
textPanel.add(textPanels[5]);

textPanels[6].add(shipMethod,FlowLayout.LEFT);
textPanel.add(textPanels[6]);

textPanels[7].add(secondAddr);
textPanel.add(textPanels[7]);

```



```

        winBox.add(labelPanel);
        winBox.add(textPanel);

        getContentPane().add(winBox, "Center");
        buttonPanel.add(ok);
        buttonPanel.add(cancel);
        ok.addActionListener(this);
        cancel.addActionListener(this);
        getContentPane().add(buttonPanel, "South");
    }

    public void actionPerformed(ActionEvent evt){

        if(evt.getSource() == ok){
            OutputStream out;
            String s;
            try{
                out = new
                    FileOutputStream("Delivery.txt", true);

                out.write(idField.getText().getBytes());
                out.write(";".getBytes());

                out.write(nameField.getText().getBytes());
                out.write(";".getBytes());

                out.write(telField.getText().getBytes());
                out.write(";".getBytes());

                out.write(alterTelField.getText().getBytes());
                out.write(";".getBytes());

                out.write(addressField.getText().getBytes());
                out.write(";".getBytes());
                s = (String)month.getSelectedItemAt();
                out.write(s.getBytes());
                out.write(" ".getBytes());

                out.write(dayField.getText().getBytes());
                out.write(",".getBytes());
                s = (String)year.getSelectedItemAt();
                out.write(s.getBytes());
                out.write(";".getBytes());
                s = (String)hour.getSelectedItemAt();
                out.write(s.getBytes());
                out.write(":".getBytes());

                out.write(minField.getText().getBytes());
                out.write(",".getBytes());
                s = (String)ampm.getSelectedItemAt();
                out.write(s.getBytes());
                out.write(";".getBytes());
                s = (String)shipMethod.getSelectedItemAt();
                out.write(s.getBytes());
                out.write(";".getBytes());
            }
        }
    }

```

```

if(!secondAddr.getText().trim().equals(""))
out.write(secondAddr.getText().getBytes());
out.write("#"+"\r").getBytes();
out.close();

}catch(IOException e){}
setVisible(false);
} //endif
else
setVisible(false);
}
public static void main(String [] args){
    JFrame f = new DeliveryFrame();
    f.show();
}

class FindCustomer implements ActionListener{
public void actionPerformed(ActionEvent evt){
    if(evt.getSource() == goButton){
        InputStream in;
        String buffer;
        String id;
        String name;
        String tel;
        String sTel;
        String addr;
        int start=0;
        int position=0;
        try{
            in = new FileInputStream("CustomerID.txt");
            BufferedReader d = new BufferedReader
                (new InputStreamReader(in));
            buffer=d.readLine();
            while(buffer!=null){
                position = buffer.indexOf(";",start);
                id = buffer.substring(start, position);
                start = position+1;

                if(idField.getText().trim().equals(id)){
                    position = buffer.indexOf(";",start);
                    name = buffer.substring(start,position);
                    nameField.setText(name);

                    start = position+1;
                    position = buffer.indexOf(";",start);
                    tel = buffer.substring(start,position);
                    telField.setText(tel);

                    start = position+1;
                    position = buffer.indexOf(";",start);

```

```

        sTel = buffer.substring(start,position);
        alterTelField.setText(sTel);

        start = position+1;
        position = buffer.indexOf("#",start);
        addr = buffer.substring(start,position);
        addressField.setText(addr);
    } //end of inner if
    buffer=d.readLine();
    start = 0;
    position = 0;
    } //end of while
    in.close();
} catch (IOException e) {}

    } //end of outer if
} //end of function
} //end of inner class
} //end of class

```

2. HTML, DHTML and Java scripting languages (as sample examples) to develop an online eBookStore prototype interface:

```

<html>
<head>
<title>eBookStore</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<!-- *****Nav Bar***** ->
<script language="javascript">
    bagPic = new Array();
    var picNo = 0;
    up =new Array();
    dn =new Array();
    ov =new Array();
    alt =new Array();
    stat =new Array();
    href =new Array();
    up[0] = "../unused.gif";
    dn[0] = "../unused.gif";
    ov[0] = "../unused.gif";
    alt[0] = "";
    stat[0] = "";
    href[0] = "#";
    up[1] = "../CS.gif";
    dn[1] = "../CSD.gif";
    ov[1] = "../CSO.gif";
    alt[1] = "go to the CS department";
    stat[1] = "view books related to computer science";
    href[1] = "../CS/CS.htm";

```

up[2] = "../ENG.gif";
dn[2] = "../ENGD.gif";
ov[2] = "../ENGO.gif";
alt[2] = "go to the engineering department";
stat[2] = "view books related to engineering";
href[2] = "../ENG/ENG.htm";
up[3] = "../MAT.gif";
dn[3] = "../MATD.gif";
ov[3] = "../MATO.gif";
alt[3] = "go to the Mathematics department";
stat[3] = "view books related to Mathematics";
href[3] = "../MAT/MAT.htm";
up[4] = "../MED.gif";
dn[4] = "../MEDD.gif";
ov[4] = "../MEDO.gif";
alt[4] = "go to the Medicine department";
stat[4] = "view books related to Medicine";
href[4] = "const.htm";
up[5] = "../BIO.gif";
dn[5] = "../BIOD.gif";
ov[5] = "../BIOO.gif";
alt[5] = "go to the Biology department";
stat[5] = "view books related to Biology";
href[5] = "const.htm";
up[6] = "../GEO.gif";
dn[6] = "../GEOD.gif";
ov[6] = "../GEOO.gif";
alt[6] = "go to the Geology department";
stat[6] = "view books related to Geology";
href[6] = "const.htm";
up[7] = "../CHE.gif";
dn[7] = "../CHED.gif";
ov[7] = "../CHEO.gif";
alt[7] = "go to the Chemistry department";
stat[7] = "view books related to Chemistry";
href[7] = "const.htm";
up[8] = "../PHY.gif";
dn[8] = "../PHYD.gif";
ov[8] = "../PHYO.gif";
alt[8] = "go to the Physics department";
stat[8] = "view books related to Physics";
href[8] = "const.htm";
up[9] = "../LAN.gif";
dn[9] = "../LAND.gif";
ov[9] = "../LANO.gif";
alt[9] = "go to the Languages department";
stat[9] = "view books related to languages";
href[9] = "const.htm";
up[10] = "../PHI.gif";
dn[10] = "../PHID.gif";

ov[10] = "../PHIO.gif";
alt[10] = "go to the Philosophy department";
stat[10] = "view books related to philosophy";
href[10] = "const.htm";
up[11] = "../HIS.gif";
dn[11] = "../HISD.gif";
ov[11] = "../HISO.gif";
alt[11] = "go to the History department";
stat[11] = "view books related to History";
href[11] = "const.htm";
up[12] = "../EDU.gif";
dn[12] = "../EDUD.gif";
ov[12] = "../EDUO.gif";
alt[12] = "go to the Education department";
stat[12] = "view books related to Education";
href[12] = "const.htm";
up[13] = "../PSY.gif";
dn[13] = "../PSYD.gif";
ov[13] = "../PSYO.gif";
alt[13] = "go to the Psychology department";
stat[13] = "view books related to Psychology";
href[13] = "const.htm";
up[14] = "../POL.gif";
dn[14] = "../POLD.gif";
ov[14] = "../POLO.gif";
alt[14] = "go to the Politics department";
stat[14] = "view books related to Politics";
href[14] = "const.htm";
up[15] = "../REL.gif";
dn[15] = "../RELD.gif";
ov[15] = "../RELO.gif";
alt[15] = "go to the Religion department";
stat[15] = "view books related to Religion";
href[15] = "const.htm";
up[16] = "../SOC.gif";
dn[16] = "../SOCD.gif";
ov[16] = "../SOCO.gif";
alt[16] = "go to the Sociology department";
stat[16] = "view books related to Sociology";
href[16] = "const.htm";
up[17] = "../LAW.gif";
dn[17] = "../LAWD.gif";
ov[17] = "../LAWO.gif";
alt[17] = "go to the Law department";
stat[17] = "view books related to law";
href[17] = "const.htm";
up[18] = "../BUS.gif";
dn[18] = "../BUSD.gif";
ov[18] = "../BUSO.gif";
alt[18] = "go to the Business department";
stat[18] = "view books related to Business";

```

href[18] = "const.htm";
up[19]  = "../ECO.gif";
dn[19]  = "../ECOD.gif";
ov[19]  = "../ECOO.gif";
alt[19] = "go to the Economics department";
stat[19] = "view books related to Economics";
href[19] = "const.htm";
up[20]  = "../ACC.gif";
dn[20]  = "../ACCD.gif";
ov[20]  = "../ACCO.gif";
alt[20] = "go to the Accountancy department";
stat[20] = "view books related to Accountancy";
href[20] = "const.htm";
up[21]  = "../ART.gif";
dn[21]  = "../ARTD.gif";
ov[21]  = "../ARTO.gif";
alt[21] = "go to the Arts department";
stat[21] = "view books related to Arts";
href[21] = "const.htm";
up[22]  = "../SPO.gif";
dn[22]  = "../SPOD.gif";
ov[22]  = "../SPOO.gif";
alt[22] = "go to the Sports department";
stat[22] = "view books related to Sports";
href[22] = "const.htm";
up[23]  = "../OTH.gif";
dn[23]  = "../OTHD.gif";
ov[23]  = "../OTHO.gif";
alt[23] = "go to others subjects";
stat[23] = "view books not related to any of the other subjects";
href[23] = "const.htm";
Tabup   = new Array();
Tabdn   = new Array();
Tabov   = new Array();
Tabalt  = new Array();
Tabstat = new Array();
Tabhref = new Array();
var curr = 0; //down is done on load
var nbBars = 8;
var nbNav = href.length/nbBars; //– 23/8;

function HH_updateTabs(event){
  var i, image, action;
  if (event && event == 'down'){
    curr=(curr+1)%nbNav;
  }else if (event && event == 'up'){
    curr=(curr-1+nbNav)%nbNav;
  }else if (event && event == 'init'){;

  }
}

```

```

        for(i=1; i<=nbBars; i++) {
            image = MM_findObj("Tab'+i);
            action = MM_findObj("Tab'+i+'a');
            if(i+(curr*nbBars)<href.length) {
                image.src = up[i+(curr*nbBars)];
                image.alt = alt[i+(curr*nbBars)];
                Tabov[i] = ov[i+(curr*nbBars)];
                Tabstat[i] = stat[i+(curr*nbBars)];
                action.href = href[i+(curr*nbBars)];
            } else { //unused
                image.src = up[0];
                image.alt = alt[0];
                Tabov[i] = ov[0];
                Tabstat[i]= stat[0];
                action.href= href[0];
            }
            image.MM_up = image.src;
        }
    }
}
//--></script>

```

<!-- *****Shelves Bar*****_

```

<script language="javascript">
    sup =new Array();
    sdn =new Array();
    sov =new Array();
    salt =new Array();
    sstat =new Array();
    shref =new Array();
    sup[0] = "../unused.gif";
    sdn[0] = "../unused.gif";
    sov[0] = "../unused.gif";
    salt[0] = "";
    sstat[0] = "";
    shref[0] = "#";
    sup[1] = "../HCI.gif";
    sdn[1] = "../HCID.gif";
    sov[1] = "../HCIO.gif";
    salt[1] = "go to the HCI department";
    sstat[1] = "view books related to HCI";
    shref[1] = "HCI/HCI.htm";
    sup[2] = "../AI.gif";
    sdn[2] = "../AID.gif";
    sov[2] = "../AIO.gif";
    salt[2] = "go to AI shelf";
    sstat[2] = "view books related to AI";
    shref[2] = "AI/AI.htm";
    sup[3] = "../OOP.gif";
    sdn[3] = "../OOPD.gif";
    sov[3] = "../OOPO.gif";

```

```

salt[3] = "go to the OOP shelf";
sstat[3] = "view books related to OOP";
shref[3] = "OOP/OOP.htm";
sup[4] = "../MED.gif";
sdn[4] = "../MEDD.gif";
sov[4] = "../MEDO.gif";
salt[4] = "go to the Medicine department";
sstat[4] = "view books related to Medicine";
shref[4] = "const.htm";
sup[5] = "../BIO.gif";
sdn[5] = "../BIOD.gif";
sov[5] = "../BIOO.gif";
salt[5] = "go to the Biology department";
sstat[5] = "view books related to Biology";
shref[5] = "const.htm";
sup[6] = "../GEO.gif";
sdn[6] = "../GEOD.gif";
sov[6] = "../GEOO.gif";
salt[6] = "go to the Geology department";
sstat[6] = "view books related to Geology";
shref[6] = "const.htm";
sup[7] = "../CHE.gif";
sdn[7] = "../CHED.gif";
sov[7] = "../CHEO.gif";
salt[7] = "go to the Chemistry department";
sstat[7] = "view books related to Chemistry";
shref[7] = "const.htm";
sup[8] = "../PHY.gif";
sdn[8] = "../PHYD.gif";
sov[8] = "../PHYO.gif";
salt[8] = "go to the Physics department";
sstat[8] = "view books related to Physics";
shref[8] = "const.htm";
sup[9] = "../LAN.gif";
sdn[9] = "../LAND.gif";
sov[9] = "../LANO.gif";
salt[9] = "go to the Languages department";
sstat[9] = "view books related to languages";
shref[9] = "const.htm";
sup[10] = "../PHI.gif";
sdn[10] = "../PHID.gif";
sov[10] = "../PHIO.gif";
salt[10] = "go to the Philosophy department";
sstat[10] = "view books related to philosophy";
shref[10] = "const.htm";
sup[11] = "../HIS.gif";
sdn[11] = "../HISD.gif";
sov[11] = "../HISO.gif";
salt[11] = "go to the History department";
sstat[11] = "view books related to History";
shref[11] = "const.htm";

```



```

sTabov = new Array();
sTabstat = new Array();
var scurr = 0; //down is done on load
var snbBars = 7;
var snbNav = shref.length/snbBars;
var oldJS = 0;
var startFlag = true;
var done = false;
function HH_updatesTabs(event){
    var i, image, action;

    if (event && event == 'left'){
//        curr=(curr+1)%nbNav;
    }else if (event && event == 'right'){
//        curr=(curr-1+nbNav)%nbNav;
    }else if (event && event == 'init'){;

    }

    for(i=1; i<=snbBars; i++) {
        image = MM_findObj('sTab'+i);
        action = MM_findObj('sTab'+i+'a');
        if(i+(curr*nbBars)<href.length) {
            image.src = sup[i+(curr*nbBars)];
            image.alt = salt[i+(curr*nbBars)];
            sTabov[i] = sov[i+(curr*nbBars)];
            sTabstat[i] = sstat[i+(curr*nbBars)];
            action.href = shref[i+(curr*nbBars)];
        } else { //unused
            image.src = sup[0];
            image.alt = salt[0];
            sTabov[i] = sov[0];
            sTabstat[i] = sstat[0];
            action.href= shref[0];
        }
    }
}

function MM_preloadImages() { //v3.0
    var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
    var i,j=d.MM_p.length,a=MM_preloadImages.arguments; for(i=0; i<a.length; i++)
    if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image; d.MM_p[j++].src=a[i];} }
}

function MM_findObj(n, d) { //v3.0
    var p,i,x; if(!d) d=document; if((p=n.indexOf("?"))>0&&parent.frames.length) {
    d=parent.frames[n.substring(p+1)].document; n=n.substring(0,p);}
    if(!(x=d[n])&&d.all) x=d.all[n]; for (i=0;!x&&i<d.forms.length;i++) x=d.forms[i][n];
    for(i=0;!x&&d.layers&&i<d.layers.length;i++) x=MM_findObj(n,d.layers[i].document);
    return x;
}

```

```

function MM_updateXY(doc,l,t,w,h)
{
  if (doc.left)
    doc.left = l;
  if (doc.top)
    doc.top = t;
  if (doc.width)
    doc.width = w;
  if (doc.height)
    doc.height = h;
}

function MM_nbGroup(event, grpName) { //v3.0
  var i,img,nbArr,args=MM_nbGroup.arguments;
  if (event == "init" && args.length > 2) {
    if ((img = MM_findObj(args[2])) != null && !img.MM_init) {
      img.MM_init = true; img.MM_up = args[3]; img.MM_dn = img.src;
      if ((nbArr = document[grpName]) == null) nbArr = document[grpName] = new Array();
      nbArr[nbArr.length] = img;
      for (i=4; i < args.length-1; i+=2) if ((img = MM_findObj(args[i])) != null) {
        if (!img.MM_up) img.MM_up = img.src;
        img.src = img.MM_dn = args[i+1];
        nbArr[nbArr.length] = img;
      }
    }
  } else if (event == "over") {
    document.MM_nbOver = nbArr = new Array();
    for (i=1; i < args.length-1; i+=3) if ((img = MM_findObj(args[i])) != null) {
      if (!img.MM_up) img.MM_up = img.src;
      img.src = (img.MM_dn && args[i+2]) ? args[i+2] : args[i+1];
      nbArr[nbArr.length] = img;
    }
  } else if (event == "out") {
    for (i=0; i < document.MM_nbOver.length; i++) {
      img = document.MM_nbOver[i]; img.src = (img.MM_dn) ? img.MM_dn : img.MM_up; }
  } else if (event == "down") {
    if ((nbArr = document[grpName]) != null)
      for (i=0; i < nbArr.length; i++) { img=nbArr[i]; img.src = img.MM_up; img.MM_dn = 0; }
    document[grpName] = nbArr = new Array();
    for (i=2; i < args.length-1; i+=2) if ((img = MM_findObj(args[i])) != null) {
      if (!img.MM_up) img.MM_up = img.src;
      img.src = img.MM_dn = args[i+1];
      nbArr[nbArr.length] = img;
    }
  }
}

function MM_showHideLayers() { //v3.0
  var i,p,v,obj,args=MM_showHideLayers.arguments;
  for (i=0; i<(args.length-2); i+=3) if ((obj=MM_findObj(args[i]))!=null) { v=args[i+2];
    if (obj.style) { obj=obj.style; v=(v=='show')?'visible':(v=='hide')?'hidden':v; }
  }
}

```

```

    obj.visibility=v; }
}

function
MM_dragLayer(objName,x,hL,hT,hW,hH,toFront,dropBack,cU,cD,cL,cR,targL,targT,tol,dropJS,
et,dragJS) { //v3.0
    //Copyright 1998 Macromedia, Inc. All rights reserved.
    var i,j,aLayer,retVal,curDrag=null,NS=(navigator.appName=='Netscape'), curLeft, curTop;
    //alert('dropJD' + dropJS);
    if (!document.all && !document.layers) return false;
    retVal = true; if(!NS && event) event.returnValue = true;
    if (MM_dragLayer.arguments.length > 1) {
        curDrag = MM_findObj(objName); if (!curDrag) return false;
        if (!document.allLayers) { document.allLayers = new Array();
            with (document) if (NS) { for (i=0; i<layers.length; i++) allLayers[i]=layers[i];
                for (i=0; i<allLayers.length; i++) if (allLayers[i].document &&
allLayers[i].document.layers)
                    with (allLayers[i].document) for (j=0; j<layers.length; j++)
allLayers[allLayers.length]=layers[j];
                } else for (i=0; i<all.length; i++) if (all[i].style&&all[i].style.position)
allLayers[allLayers.length]=all[i];}
        curDrag.MM_dragOk=true; curDrag.MM_targL=targL; curDrag.MM_targT=targT;
        curDrag.MM_tol=Math.pow(tol,2); curDrag.MM_hLeft=hL; curDrag.MM_hTop=hT;
        curDrag.MM_hWidth=hW; curDrag.MM_hHeight=hH; curDrag.MM_toFront=toFront;
        curDrag.MM_dropBack=dropBack; curDrag.MM_dropJS=dropJS;
        curDrag.MM_everyTime=et; curDrag.MM_dragJS=dragJS;
        curDrag.MM_oldZ = (NS)?curDrag.zIndex:curDrag.style.zIndex;
        curLeft= (NS)?curDrag.left:curDrag.style.pixelLeft; curDrag.MM_startL = curLeft;
        curTop = (NS)?curDrag.top:curDrag.style.pixelTop; curDrag.MM_startT = curTop;
        curDrag.MM_bL=(cL<0)?null:curLeft-cL; curDrag.MM_bT=(cU<0)?null:curTop -cU;
        curDrag.MM_bR=(cR<0)?null:curLeft+cR; curDrag.MM_bB=(cD<0)?null:curTop +cD;
        curDrag.MM_LEFTRIGHT=0; curDrag.MM_UPDOWN=0; curDrag.MM_SNAPPED=false;
//use in your JS!
        document.onmousedown = MM_dragLayer; document.onmouseup = MM_dragLayer;
        if (NS) document.captureEvents(Event.MOUSEDOWN|Event.MOUSEUP);
    } else {
        var theEvent = ((NS)?objName.type:event.type);
        if (theEvent == 'mousedown') {
            var mouseX = (NS)?objName.pageX : event.clientX + document.body.scrollLeft;
            var mouseY = (NS)?objName.pageY : event.clientY + document.body.scrollTop;
            var maxDragZ=null; document.MM_maxZ = 0;
            for (i=0; i<document.allLayers.length; i++) { aLayer = document.allLayers[i];
                var aLayerZ = (NS)?aLayer.zIndex:aLayer.style.zIndex;
                if (aLayerZ > document.MM_maxZ) document.MM_maxZ = aLayerZ;
                var isVisible = (((NS)?aLayer.visibility:aLayer.style.visibility).indexOf('hid') == -1);
                if (aLayer.MM_dragOk != null && isVisible) with (aLayer) {
                    var parentL=0; var parentT=0;
                    if (!NS) { parentLayer = aLayer.parentElement;
                        while (parentLayer != null && parentLayer.style.position) {
                            parentL += parentLayer.offsetLeft; parentT += parentLayer.offsetTop;
                            parentLayer = parentLayer.parentElement; } }

```

```

var tmpX=mouseX-(((NS)?pageX:style.pixelLeft+parentL)+MM_hLeft);
var tmpY=mouseY-(((NS)?pageY:style.pixelTop +parentT)+MM_hTop);
var tmpW = MM_hWidth; if (tmpW <= 0) tmpW += ((NS)?clip.width :offsetWidth);
var tmpH = MM_hHeight; if (tmpH <= 0) tmpH += ((NS)?clip.height:offsetHeight);
if ((0 <= tmpX && tmpX < tmpW && 0 <= tmpY && tmpY < tmpH) &&
    (maxDragZ == null
     || maxDragZ <= aLayerZ)) { curDrag = aLayer; maxDragZ = aLayerZ; } }
if (curDrag) {
    document.onmousemove = MM_dragLayer; if (NS)
document.captureEvents(Event.MOUSEMOVE);
    curLeft = (NS)?curDrag.left:curDrag.style.pixelLeft;
    curTop = (NS)?curDrag.top:curDrag.style.pixelTop;
    MM_oldX = mouseX - curLeft; MM_oldY = mouseY - curTop;
    document.MM_curDrag = curDrag; curDrag.MM_SNAPPED=false;
    if(curDrag.MM_toFront) {
        eval('curDrag.'+(NS)?":style."+'zIndex=document.MM_maxZ+1');
        if (!curDrag.MM_dropBack) document.MM_maxZ++; }
    retVal = false; if(!NS) event.returnValue = false;
} } else if (theEvent == 'mousemove') {
    if (document.MM_curDrag) with (document.MM_curDrag) {
        var mouseX = (NS)?objName.pageX : event.clientX + document.body.scrollLeft;
        var mouseY = (NS)?objName.pageY : event.clientY + document.body.scrollTop;
        newLeft = mouseX-MM_oldX; newTop = mouseY-MM_oldY;
        if (MM_bL!=null) newLeft = Math.max(newLeft,MM_bL);
        if (MM_bR!=null) newLeft = Math.min(newLeft,MM_bR);
        if (MM_bT!=null) newTop = Math.max(newTop ,MM_bT);
        if (MM_bB!=null) newTop = Math.min(newTop ,MM_bB);
        MM_LEFTRIGHT = newLeft-MM_startL; MM_UPDOWN = newTop-MM_startT;
        if (NS) {left = newLeft; top = newTop;}
        else {style.pixelLeft = newLeft; style.pixelTop = newTop;}
        if (MM_dragJS) eval(MM_dragJS);
        retVal = false; if(!NS) event.returnValue = false;
} } else if (theEvent == 'mouseup') {
    document.onmousemove = null;
    if (NS) document.releaseEvents(Event.MOUSEMOVE);
    if (NS) document.captureEvents(Event.MOUSEDOWN); //for mac NS
    if (document.MM_curDrag) with (document.MM_curDrag) {
        //alert('style.pixelLeft = ' + style.pixelLeft);
        if (typeof MM_targL == 'number' && typeof MM_targT == 'number' &&
            (Math.pow(MM_targL-((NS)?left:style.pixelLeft),2)+
             Math.pow(MM_targT-((NS)?top:style.pixelTop),2))<=MM_tol) {
            if (NS) {left = MM_targL; top = MM_targT;}
            else {style.pixelLeft = MM_targL; style.pixelTop = MM_targT;}
            MM_SNAPPED = true; MM_LEFTRIGHT = MM_startL-MM_targL; MM_UPDOWN =
MM_startT-MM_targT; }
        if (MM_everyTime || MM_SNAPPED)
        {
            //if (MM_SNAPPED)
            // alert('MM_SNAP');
            //if (MM_everyTime)
            // alert('MM_every');

```

```
    eval(MM_dropJS);
  }
  if(MM_dropBack) {if (NS) zIndex = MM_oldZ; else style.zIndex = MM_oldZ;}
  retVal = false; if(!NS) event.returnValue = false; }
  document.MM_curDrag = null;
}
if (NS) document.routeEvent(objName);
}
return retVal;
}
```

```
{staff deleted}
```

```
<p>&nbsp;</p>
```

```
<hr size="2">
<p><font size="2"> problems? contact <a
href="mailto:razibul@cs.concordia.ca">webmaster<br>
</a>last modification: 30 April, 2000 </font></p>
</td>
</tr>
</table>
</body>
</html>
```

Reference

- 1- Myers, B.A. User Interface Software Tools. ACM Transactions on Computer Human Interaction 2(1), 1995, pp. 64-103.
- 2- Nielsen J. Designing Web Usability: The Practice of Simplicity. New Riders Publishing 1999
- 3- Norman, D.A. The Design of Everyday Things. 1990
- 4- Rader, C., G. Cherry, C. Brand, A. Repining and C. Lewis. Designing Mixed Textual and Iconic Programming Languages for Novice Users. http://www.cs.colorado.edu/homes/crader/public_html/VL98/VL98.html
- 5- Smith, D.C., A. Cypher and L. Tesler. Novice Programming Comes of Age. In Communications of the ACM, 43(3), March 2000, pp. 75-81.
- 6- Walrath, K., Campione, M. The JFC Swing Tutorial: A Guide to Constructing GUIs. Addison-Wesley Pub Co. 1999
- 7- Robert M. Mulligan, Mark W. Altom and David K. Simkin, "User interface design in the trenches: some tips on shooting from the hip" <http://www.acm.org/pubs/articles/proceedings/chi/108844/p232mulligan/p232-mulligan.pdf>
- 8- John Mullaly, "IBM RealThings" <http://www.acm.org/pubs/articles/proceedings/chi/286498/p13-mullaly/p13-mullaly.pdf>
- 9- Brad A. Myers, Richard G. McDaniel, Robert C. Miller, Alan S. Ferreny, Andrew Faulring, Bruce D. Kyle, Andrew Mickish, Alex Klimovitski and Patrick Doane. "The Amulet Environment: New Models for Effective User Interface Software Development," IEEE Transactions on Software Engineering, Vol. 23, no. 6. June, 1997. pp. 347-365. http://www.artis.uni-oldenburg.de/Books/Java_GuideLines/higc.htm
- 10-[WebSite] See the E-CO System Project website at <http://eco.eit.com>
- 11-G. Booch, J. Rumbaugh, I. Jacobson "The Unified Modelling Language User Guide, Addison Wesley Pub Co. 1999
- 12-Martin Fowler, Kendall Scott, UML Distilled-Appling the standard Object modeling language, Addison-Wesley Pub Co. 1997
- 13-Stefano Ceri & Piero Fraternali's WebML: a modeling Language and Tool Suite <http://www.webml.org>
- 14-IBM's Websphere Commerce Server – Service Provider Edition <http://www.ibm.software.com>
- 15-Mercantec's Softcart Solution <http://www.mercantec.com/solutions/index.html>
Intershop - <http://www.intershop.com>
- 16-Macworld, Web Publisher's Essential Tool Kit <http://bondiboard.macpublishing.net/1998/05/features/4282.html>
- 17-Creative Good, <http://www.creativegood.com/>
<http://www.creativegood.com/creativegood-whitepaper.pdf>