Aspect-Oriented Development of an E-Commerce Application

Doğan Altunbay, Damla Arifoglu, Hilal Zitouni

Department of Computer Engineering, Bilkent University

Bilkent, 06800, Ankara, Turkey

(altunbay, arifoglu,zitouni)@cs.bilkent.edu.tr

Abstract

E-commerce applications which are used for selling and buying products on electronic platforms have become popular with the recent growth in internet usage. Different kinds of applications for e-commerce applications are developed considering the concerns of shipment, financing persistency and ordering systems. However; modularization of these concerns are not usually easy with object-oriented software development. AOSD is a recently developed new software development technology which provides abstraction mechanism for these so-called crosscutting concerns. In this paper, we will be discussing an aspect-oriented software development of an e-commerce bookstore application and we will briefly give the lessons learned throughout the development phase.

Keywords
E-commerce systems, Aspect-oriented Programming, Exceptions, User Authentication, , Design Patterns,

1. INTRODUCTION

Separation of concerns is one of the most important requirements of good system design and implementation, since it leads to maintainability, extendibility, reusability, adaptability and clarity which can all be achieved by the help of Aspect-oriented Systems. AOP (Aspect-oriented Programming) deals with crosscutting concerns in a modular way and this helps modularity, easier developing of the code and increase of reusability.

E-commerce systems are online systems, which sell products or provide services via internet and having an e-commerce website will increase the benefit of the company since web based communication and shipment will be easier for the clients from the other sides of the world. Some of the examples for e-commerce applications are [4, 8, 9] In this paper, we use a case study named "e-commerce bookstore system" in aspect-oriented development in order to show the benefits of aspect-oriented design over object oriented design. Our bookstore application is similar to e-bay which is a worldwide famous e-commerce internet web site.

Our system, which is an e-commerce book store application, originally provides users to search, and order his selected choice of the documents that are retrieved from the database. The paper will more focus on the implementation of the application itself rather than the server – client part since the server – client part of the application is not the basic focus of how usage of aspect-oriented development has improved our previous, original e-commerce bookstore application.

The previous system was actually did not include the authentication for different user types. In other words, all the users who enter the site may search through the document database; however the non-registered users should not be able to order any selected document. On the contrary, the system was not checking whether or not to allow users to give orders. The aspect-oriented development on the other hand will handle to improve the previous system by providing the users different types of authentications.

Applying the aspect-oriented development on the previously implemented code for improvement has proved its superiority over changing the system manually from the code inside.
The organization of the paper is as follows: in section 1, a brief information on the e-commerce application concerns will be given, then in section 3 our e-commerce case study and its detailed design will be explained in object oriented design, and then the improvements provided via aspect-oriented design will be mentioned. This will be followed by the aspects we have implemented under section 4, *Aspect-oriented Programming*. Afterwards, in section 5 the related works achieved up to now will be given, and finally we will give the discussion and conclusion in section 6 and.

2. OBJECT-ORIENTED DESIGN

In this section an example of the design of e-commerce application system will be introduced. In the subsequent sections the application will be improved with the help of aspect-oriented development.

An e-commerce application is used for buying and selling products over electronic systems like internet. Our book store application is an e-commerce application in which people can buy and search documents from a database of documents over internet. In our application we have not implemented the server client part since it is not really the focus of applying aspect-oriented development on a system. Therefore, we will give brief information on how the bookstore application is implemented without the internet part and the UML modelling of our system.

Here we will introduce a simple book store application in which the users will search through the document database of the Book Store, in order to view and/or order documents. In this system the database has three types of documents which are Book, Magazine and Journal, each of which has soft and hard copy versions.

The program is implemented by using some of the design patterns to overcome the problems that might occur. In order to explain the system better, brief information of the design is given along with the corresponding UML diagrams and screenshots.

Once the user is logged in, the main user interface of the application which is given in Figure 1 will be seen.

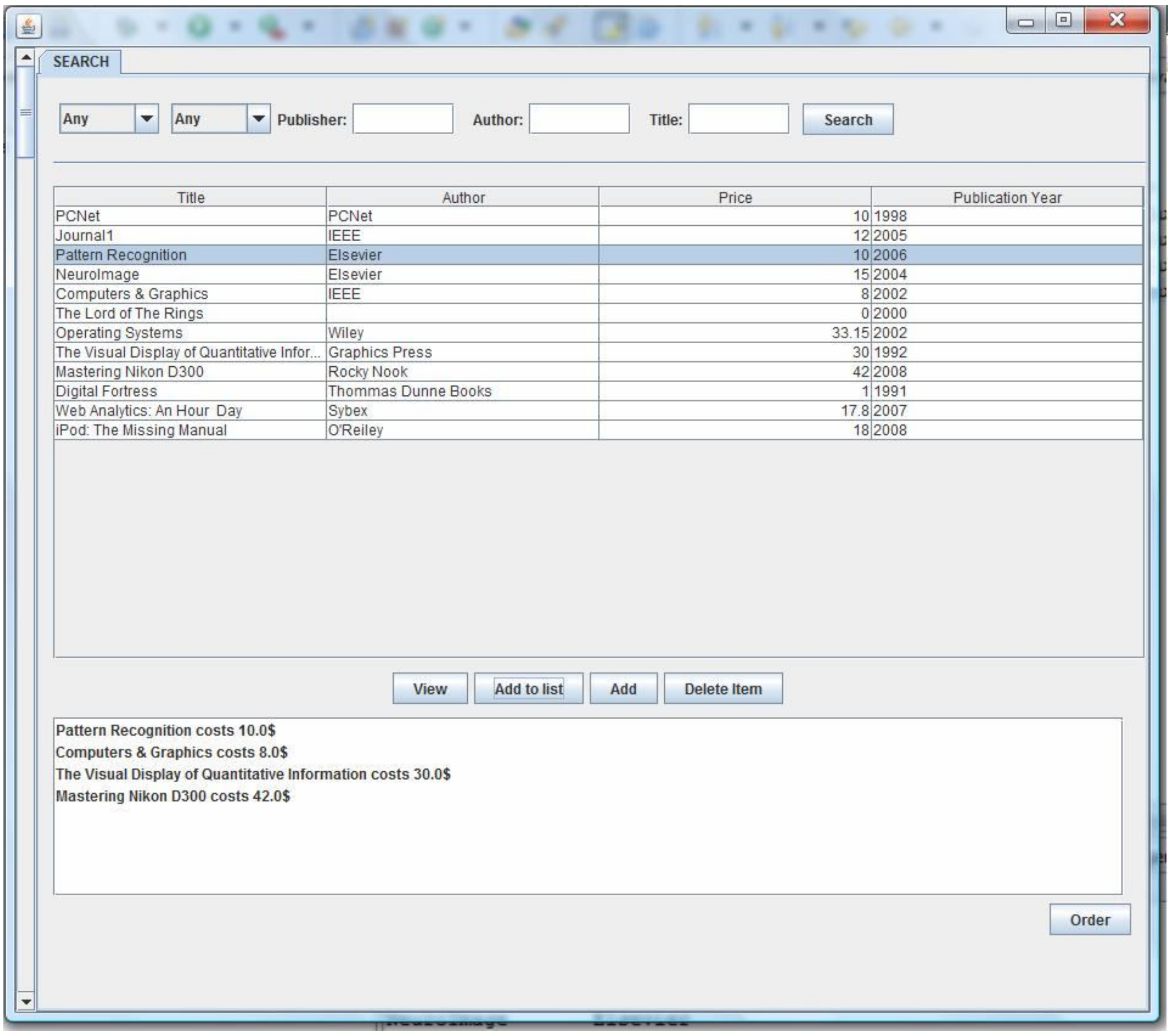
At the top of the screen user will search through the database filling specific fields and pressing “Search” button, then the corresponding document list will appear on the list. Once the documents are listed user may either select a document and view its information pressing the “View” button, or select several documents and add them to the list below using the “Add To List” button, which will later be the list of orders once the “Order” button is pressed. Pressing the “Order” button another window specifying the prices, quantity and the total price of the listed orders will come up and will offer user to enter his address and his country for shipment information.

2.1. Design Patterns Used and UML Diagrams

2.1.1. Overall Package System

Before moving to design patterns, below there is the UML diagram for overall packages of the system.

The system has four different packages, which are ***viewer***, ***item***, ***store*** and ***user,*** in its implementation. The ***viewer*** package holds the classes for viewing operation for each type of document that is selected from the list; ***order*** package holds the classes about the ordering and shipment. The user package has user related classes, and ***store*** is a helper for the



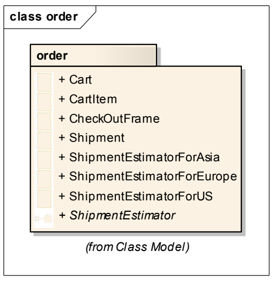
Title	Author	Price	Publication Year
PCNet	PCNet	10	1998
Journal1	IEEE	12	2005
Pattern Recognition	Elsevier	10	2006
NeuroImage	Elsevier	15	2004
Computers & Graphics	IEEE	8	2002
The Lord of The Rings		0	2000
Operating Systems	Wiley	33.15	2002
The Visual Display of Quantitative Infor...	Graphics Press	30	1992
Mastering Nikon D300	Rocky Nook	42	2008
Digital Fortress	Thommas Dunne Books	1	1991
Web Analytics: An Hour Day	Sybex	17.8	2007
iPod: The Missing Manual	O'Reiley	18	2008

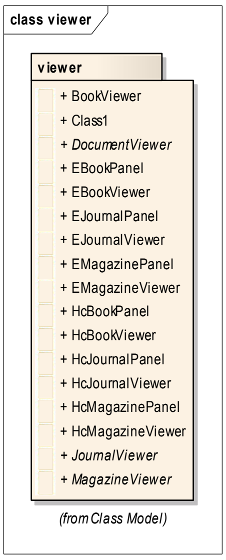
Pattern Recognition costs 10.0$
Computers & Graphics costs 8.0$
The Visual Display of Quantitative Information costs 30.0$
Mastering Nikon D300 costs 42.0$

Figure 1 : The main user interface of the application after user logged in

viewing operation, finally, the package ***item*** has the book store item types.

In Figure 2 the user package has different types of users which are implemented after aspect development for improvement.





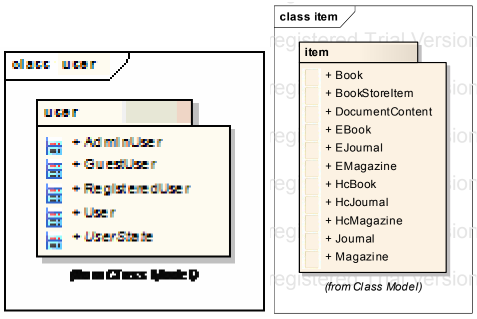


Figure 2 : Order Package and Viewer Package, User Package, Item Package

2.1.2. Design Patterns

Three different design patterns are implemented for the BookStore Application; Abstract Factory Method, Strategy Method and State Method. The first two design patterns are developed in the original BookStore Application, whereas the last one, State Method, is added after the production aspects are added to improve the system.

2.1.2.1. Abstract Factory Design Pattern

The Abstract Factory is a creational design pattern that is used for grouping the object factories of a common theme. The factory will decide the concrete object type of its abstract object to be created. In this application, this design pattern is implemented for the view operation of documents. In other words, we will have a DocumentViewerFactory which will select among the soft and hardcopy versions of document types, Book, Magazine and Journal. The abstract class DocumentViewerFactory, will decide on which of the six different types of documents will be displayed on the common panel of the dialog box that comes up when the "View" button is pressed for the corresponding selected document item, from the table. The UML diagrams for the viewer and store packages are given in Figure 3

2.1.2.2. Strategy Method Design Pattern

The Strategy Design Pattern is a behavioural design pattern that is used for the differentiation between the different implementation of same operations. In other words, same operations differ for different types of objects in implementation, and strategy pattern handles this by implementing the different implementations of same operations in subclasses. Here, we had our *shipment* procedures using strategy design pattern. For shipment we have a Shipment class which uses a ShipmentEstimator as an abstract class which has subclasses of different shipment estimator classes for different countries. The UML diagram for the corresponding system is given in Figure 4.

2.1.2.3. State Method Design Pattern

The State Design Pattern is a behavioural design pattern which is used for changing the state of an object at run time. Here, the state design pattern is used for changing the state of the user one of the three states; Guest, Registered and Admin User. To give an example of a state change; we can give the scenario of a guest user who tries to give an order. When a guest user presses the *Order*

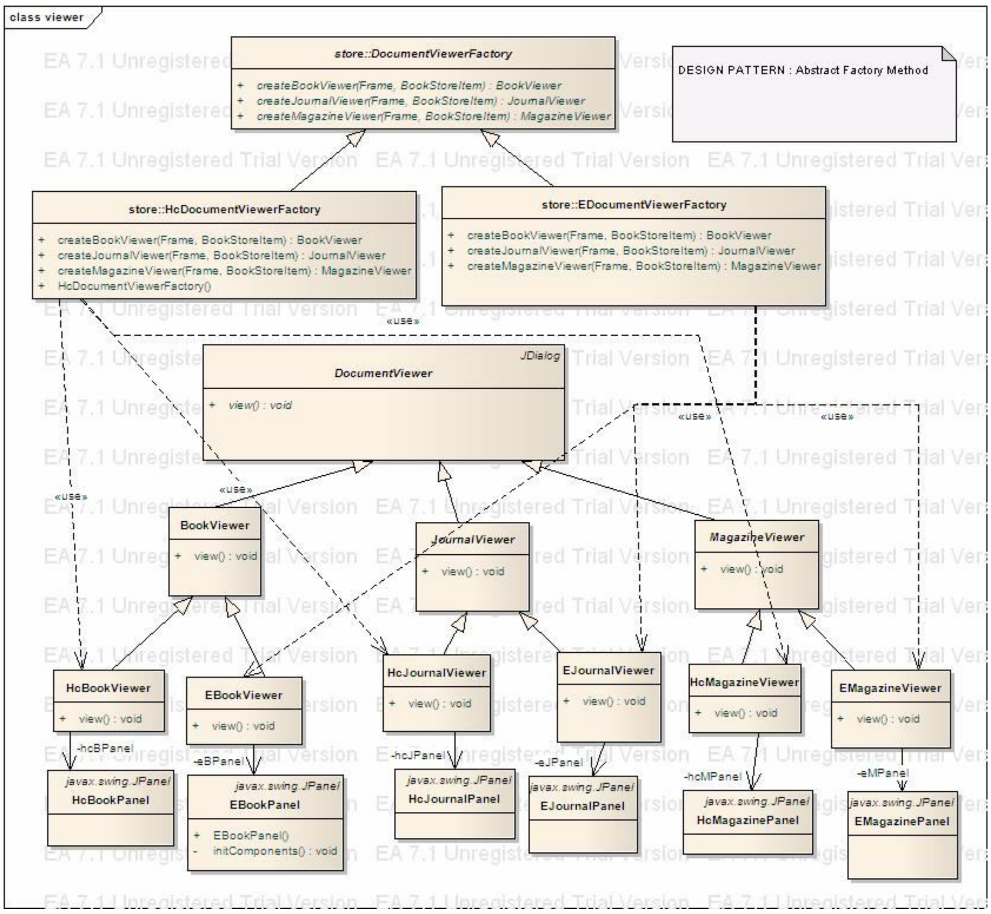

class viewer
store::DocumentViewerFactory
+ createBookViewer(Frame, BookStoreItem) : BookViewer
+ createJournalViewer(Frame, BookStoreItem) : JournalViewer
+ createMagazineViewer(Frame, BookStoreItem) : MagazineViewer
DESIGN PATTERN : Abstract Factory Method
store::HcDocumentViewerFactory
+ createBookViewer(Frame, BookStoreItem) : BookViewer
+ createJournalViewer(Frame, BookStoreItem) : JournalViewer
+ createMagazineViewer(Frame, BookStoreItem) : MagazineViewer
+ HcDocumentViewerFactory()
store::EDocumentViewerFactory
+ createBookViewer(Frame, BookStoreItem) : BookViewer
+ createJournalViewer(Frame, BookStoreItem) : JournalViewer
+ createMagazineViewer(Frame, BookStoreItem) : MagazineViewer
«use»
JDialog
DocumentViewer
+ view() : void
«use»
«use»
«use»
«use»
BookViewer
+ view() : void
JournalViewer
+ view() : void
MagazineViewer
+ view() : void
HcBookViewer
+ view() : void
EBookViewer
+ view() : void
HcJournalViewer
+ view() : void
EJournalViewer
+ view() : void
HcMagazineViewer
+ view() : void
EMagazineViewer
+ view() : void
-hcBPanel
-eBPanel
-hcJPanel
-eJPanel
-hcMPanel
-eMPanel
javax.swing.JPanel
HcBookPanel
javax.swing.JPanel
EBookPanel
+ EBookPanel()
- initComponents() : void
javax.swing.JPanel
HcJournalPanel
javax.swing.JPanel
EJournalPanel
javax.swing.JPanel
HcMagazinePanel
javax.swing.JPanel
EMagazinePanel

Figure 3 - Abstract Factory Design Pattern

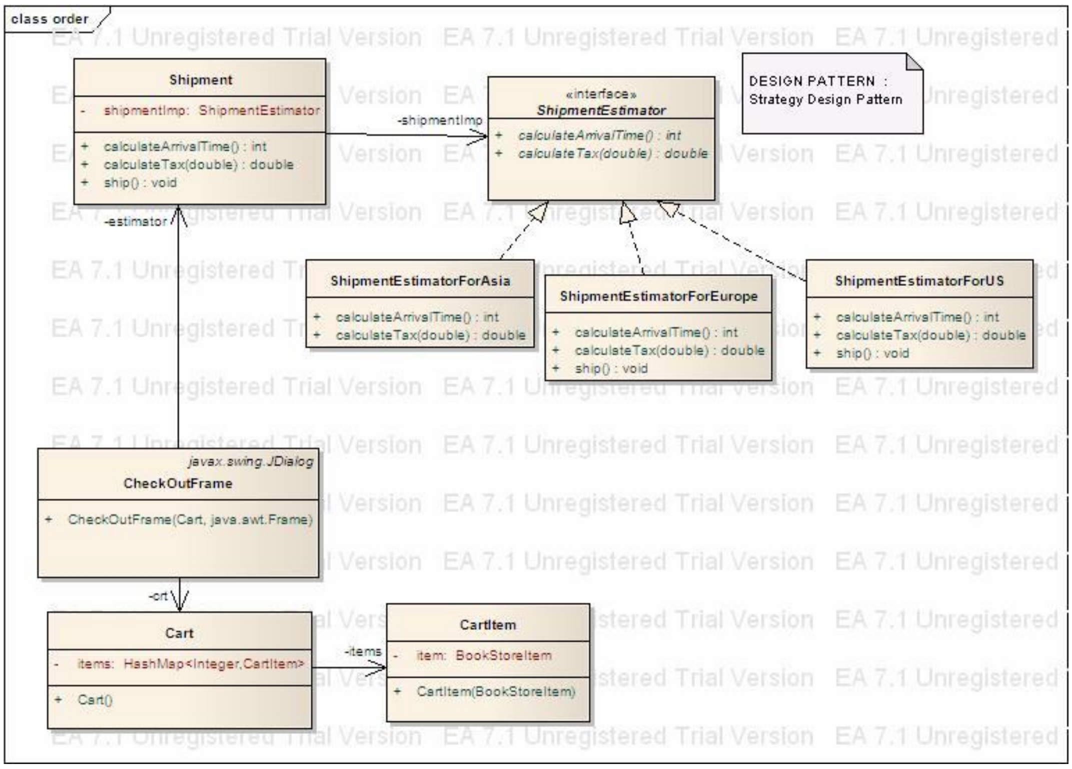

class order
Shipment
- shipmentImp: ShipmentEstimator
+ calculateArrivalTime() : int
+ calculateTax(double) : double
+ ship() : void
-shipmentImp
«interface»
ShipmentEstimator
+ calculateArrivalTime() : int
+ calculateTax(double) : double
DESIGN PATTERN : Strategy Design Pattern
-estimator
ShipmentEstimatorForAsia
+ calculateArrivalTime() : int
+ calculateTax(double) : double
ShipmentEstimatorForEurope
+ calculateArrivalTime() : int
+ calculateTax(double) : double
+ ship() : void
ShipmentEstimatorForUS
+ calculateArrivalTime() : int
+ calculateTax(double) : double
+ ship() : void
javax.swing.JDialog
CheckOutFrame
+ CheckOutFrame(Cart, java.awt.Frame)
-crt
Cart
- items: HashMap<Integer,CartItem>
+ Cart()
-items
CartItem
- item: BookStoreItem
+ CartItem(BookStoreItem)

Figure 4 : Strategy Design Pattern for Shipment

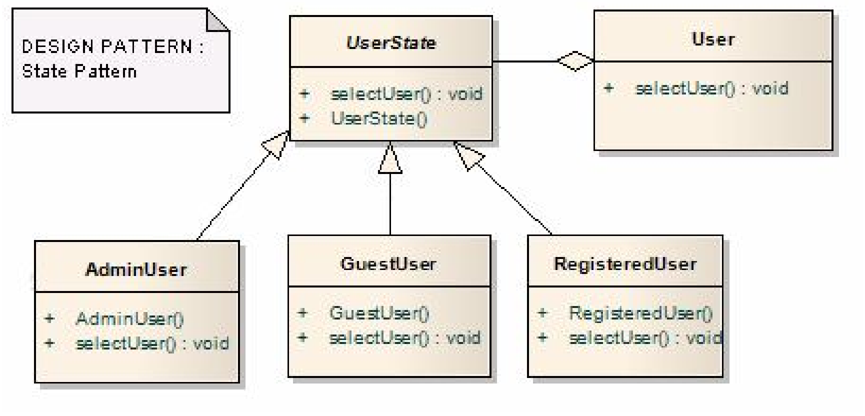


Figure 5 – State Design Pattern for User State

Button a registration user interface will come up and the filling the form he can change his state from *guest* to *registered*, and become a registered user. After changing his state he may give order as he wants from the main user interface. The corresponding UML diagram is given in Figure 5.

3. CROSSCUTTING CONCERNS

E-Commerce Systems are electronic applications that provide users to sell and buy products throughout the world especially via internet [10]. In an e-commerce application the shipment, financial persistency and security of the personal information of the users are some of the vital concerns.

In our e-commerce bookstore application, we noticed that improving the system will result in some of the concerns to be scattered over multiple classes and crosscutting of the basic functionality of the code in these classes. Object oriented programming cannot handle these crosscutting concerns to such an extent that Aspect-oriented Software development can. Therefore, we have especially focused on the concerns of the e-commerce applications. The concerns of our e-commerce book store application can be listed as follows:

- Authorization and Authentication
- Null Checking
- Adding new functionalities to the system as the applications grows with the need of the customers.

Let us give wider information on these concerns.

3.1. Authentication and Authorization Concern

User authentication and authorization concerns are related to determining who is allowed to perform which operations. We need to ensure that users are authenticated to do certain operations. Therefore, authentication is the process of determining whether someone is who he declares to be.

Unauthenticated users have accesses to web pages which do not have any authentication policies. For the authors of [2] authentication is vitally important, since a web application can be a piece of a cake for an intruder, and needs to be protected.

In our e-commerce bookstore system, we used development aspects which will be explained in section 5 to handle authentication concerns. Authorization, on the other hand, is the process of deciding on the permissions of a user over the system. For example in our system there are three types of users, Admin User, Guest

User and Registered User, each of which have different authorizations. A Guest User cannot do anything but the search operation. A Registered User can search, order and view the documents in the database. Admin User on the other hand has all three accesses on the system, plus he can add/delete items from/to system. Our aspects for authorization handle deciding on the accesses to be provided to a user.

3.2. Null Check

In our e-commerce bookstore application system, we use a database to hold the book items (such as magazines, journals, books and their online versions). Null checking will control the problems that might occur about the database.

3.3. **Adding New Functionalities**

Since e-commerce applications give services to people from all over the world by being an online application, user needs and new demands will come up with the increasing amount in the usage of the e-commerce applications. Therefore, new functionalities and new objects may be needed to be added to the system. In our e-commerce application, at first, every user has the same authentication regardless of whether they should be a guest user, an admin user or a registered user and any user could reach the same user interface. However, with aspect-oriented development, in order to provide different authentication rights to the users, new types of users; such as Admin Users, Registered Users and Guest Users, are introduced to the system.

4. ASPECT-ORIENTED PROGRAMMING

In the previous section we defined our concerns that are scattered over multiple classes. Scattering results in low cohesion and high coupling which is an important weakness for software development. Aspect-oriented programming is a useful method for solving these problems.

With the help of AOP, developers can produce far less complex software systems. AOP not only provides effective mechanisms at the development stage, but also provides easiness in maintaining and upgrading the system. Below we discuss our solution to the crosscutting concerns described in section 2.

4.1. Handling Aspects

In our work, we take the advantage of JBoss AOP for aspect-oriented programming. JBoss AOP is an AOP framework which provides pure java syntax to programmers. In JBoss AOP, aspect weaving is performed via reflection with the help of xml files that define the bindings between join points and advices.

As defined in the previous sections, our case study involves a number of crosscutting concerns. The following sub sections describe our aspect-oriented solutions which address these problems.

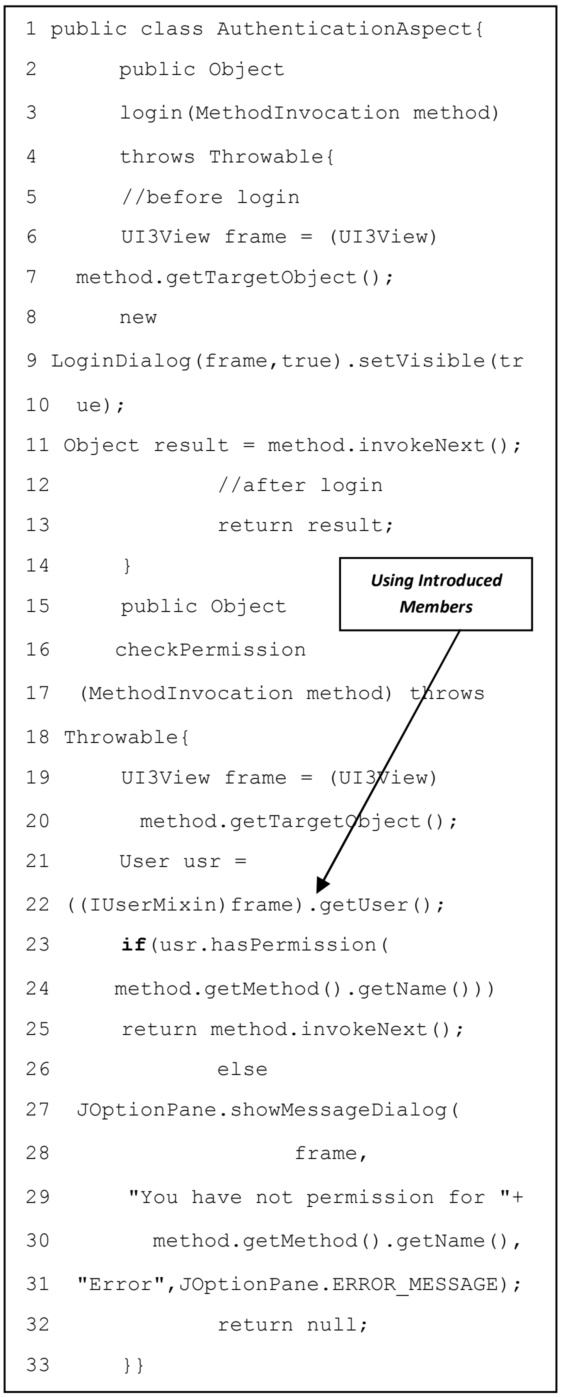
```
private void viewDoc(BookStoreItem item){
      if(!this.user.isAuthenticated())
      throw new

      UnsupportedOperationException();

      DocumentViewer viewer = null;
      .
      .
      .
      viewer=
  docFactory.createMagazineViewer(this,item);

      viewer.view();
  }
```

Figure 6 – Example insertion of authentication related code to legacy code

4.1.1. Authentication and Authorization

Authentication and authorization of users have vital importance for e-commerce applications. It is necessary to be able identify users and control their actions while interacting with the system.

Authentication and authorization of users have vital importance for e-commerce applications. It is necessary to be able identify users and control their actions while interacting with the system.



```
public class AuthenticationAspect{
     public Object
     login(MethodInvocation method)
     throws Throwable{
     //before login
     UI3View frame = (UI3View)
  method.getTargetObject();
     new
LoginDialog(frame,true).setVisible(true);
 Object result = method.invokeNext();
             //after login
             return result;
     }
     public Object
     checkPermission
  (MethodInvocation method) throws
 Throwable{
     UI3View frame = (UI3View)
       method.getTargetObject();
     User usr =
 ((IUserMixin)frame).getUser();
     if(usr.hasPermission(
     method.getMethod().getName()))
     return method.invokeNext();
             else
  JOptionPane.showMessageDialog(
                   frame,
      "You have not permission for "+
        method.getMethod().getName(),
  "Error",JOptionPane.ERROR_MESSAGE);
             return null;
     }}
```

Figure 7 – Authentication Aspect

For our case study, we would prefer the users to be able to browse the documents in the system without any restrictions. However, if a user wants to view a document's soft copy, order an item, add an item or delete an item the system should confirm the identity of the user.

In order to handle these requirements, we have to make modifications on our legacy code. Thus, we have to add a bunch of code into the action handlers of the "View" button, "Order" button, "Add" button, and "Delete" button that prompts user to login to system (see Figure 6). This kind of approach leads multiple code segments related with one single concern. Therefore dealing with and tracking these concerns becomes inefficient.

However, we can solve the scattering problem using AOP. Then, we define an aspect, called "*AuthenticationAspect*" (see Figure 7) that gathers authentication and authorization related procedures. The AuthenticationAspect introduces two methods; "*login*" for authenticating the user, and "*checkPermission*" for authorizing the user.

To bind these advice methods to action handlers mentioned above, we need to specify them in the jboss-aop.xml file (see Figure 8).

This file defines the aspects, advices, and pointcuts for the use of the framework. Aspect weaving is performed via reflection in accordance with the definitions in this file.

Another issue in this solution is that we have to add a private field of class User to the application in order to store authentication and authorization information. AOP paradigm provides introduction mechanism which allows programmers to extend existing classes and add new members to them.

```
<aop>
  <pointcut expr="call(public void
UI3View-
&gt;display())" name="show"/>
 <pointcut expr="execution(private void
ui.UI3View&gt;
viewDoc(item.BookStoreItem))"
name="view"/>
   <aspect
class="aspect.AuthenticationAspect"
scope="PER_VM"/>
   <bind pointcut="show">
       <advice
aspect="aspect.AuthenticationAspect"
name="log in"/>
   </bind>
   <bind pointcut="view">
       <advice
spect="aspect.AuthenticationAspect"
 name="checkPermission"/>
  </bind>
  <bind pointcut="execution(public void
ui.UI3View-&gt;display())">
       <advice
aspect="aspect.AuthenticationAspect"
name="login"/>
  </bind>
   <introduction class="ui.UI3View">
      <mixin>
  <interfaces>
user.IUserMixin</interfaces>
           <class>user.UserMixin</class>
           <construction>new
user.UserMixin()</construction>
       </mixin>
   </introduction>
</aop>
```

Figure 8 – Sample jboss-aop.xml file.

In JBoss AOP, introductions are done using mixin mechanism. As the name reflects, mixin mechanism uses a mixed data structure for each member introduction to a class. In order to add a field to a class, we first define an interface that includes the methods which accesses that field. Then we place the implementation of the interface which will be mixed with the target class of introduction in runtime. After that, we need to specify how these two classes will be mixed in jboss-aop.xml (see listing 3). Then we can access the field within several places of the code.

4.1.2. Exception Handling:

Exception Handling is used for handling the problems that might occur during the execution of a program. In object oriented programming, exception handlers are implemented for each and every point that an exception might occur using the try and catch code blocks. Implementing exception handlers with OOD not only complicates coding but also makes the code scattered. Moreover, implementing new exception handlers to the system causes a lot of code blocks to be added, which means huge amount of intervention to the code. Aspect-Oriented Development on the other hand, saves programmers from changing and adding code blocks to such an extent. Rather programmers will easily implement advices for the exception handlers in an aspect, and will use them by binding with the specified pointcuts which are the places for the handler advices to be invoked.

Besides, implementing all the exception handlers in a separate aspect code will increase modularity. Rather than having exception handler codes all over the program, it will be coherently collected in a separate part.

4.1.3. Enforcement Aspects:

In our project we needed to use enforcement aspects for restricting the invocation of some of the methods from certain classes. For example, in our program, in order to view the information of the selected document, user will press the “View” button after selecting a document from the list. However, it should be under control that no programmers should ever invoke the *getContent()* method for any of the six different panels, other than the *EDocumentViewerFactory* class. This is because the content field information of a document should only be available for softcopy versions of the documents, and since the softcopy versions are viewed through *EDocumentViewerFactory* it should not be allowed to be invoked from somewhere else. In case of such a mistake, AOD will handle preventing those methods to be invoked from some place else. However, in JBoss, contrary to aspectj, implementing enforcements and controlling the invocation during compile time is not handled. Therefore, in order to provide this opportunity of control over method calls, we have implemented advices which will prevent the invocation of those methods during run time.

4.1.4. Null Check

As it is explained in section 3, null check will check on the database problems. In Database Systems it is often possible for a primary field to be null, which will arise problems for the insertion to the data. In order not to cause any problems during insertion operations, null check should be done before execution of an insert statement. This is the case where Null Check concern arises in our application. Before invoking the addItem operation of the DBManager class, a null check is done.

5. RELATED WORK

There are some previous works that compares the aspect-oriented software development and object oriented software development in web applications. One of these studies, is Reina, A. et al [2]. To show the advantages of the aspect-oriented development in web applications, they developed an application using AspectJ and Java Server Pages (JSP) and Cocoon, which is a web framework based on XML to publish web pages. The work in this paper will not be discussed in detail, however the two common concerns that they consider will be explained shortly. These common two concerns are similar to the ones that are in our e-commerce web application. First one is security (authentication) and the second one is exception handling. On the other hand, they also address some different concerns from ours which are Pooling, Caching and Logging concerns. They handle authentication concern by using Cocoon to publish web pages and Cocoon itself already handles authentication concern via a module. In Cocoon, the authentication handler is an object that controls the access to the resources. Each resource can be related to one authentication handler and one authentication handler manages the access rights to all the resources that are related to it. In this system, you need a username and a password to access the main page and if the user isn't authenticated, the login page will be shown. Also, they handle another concern which is "Design by Contact" and they state that the implementation of this concern using AspectJ is reduced. The NullContract aspect is pointcut which picks an execution of all the database methods. If any null parameter is found, then an IllegalArgumentException is thrown. Also, Cocoon itself has a form-validator, in which the characteristics of the parameters of the form are described. A property named nullable is used and indicates that the parameter can not be null, so Design by Contact concern is handled.

6. DISCUSSION

Using aspect-oriented software on the projects that are designed carefully and which obeys the code standards is a real advantage on improving the system with less code generation and more reusability. It is a fact that making a small change in the system requires a lot of changes in the whole project.

Programmer should add or remove code from a numerous class. However using aspect-oriented development saves programmer from intervening into code too much, rather, writing aspect codes for the corresponding changes is sufficient. However, since aspect-oriented development is a new technology in programming, programmers do not yet implement in a way to make the usage of aspect easier. Writing aspect codes for previously implemented programs will be harder in the sense of not using standard naming for the classes or methods. On the other hand, knowing a program can be improved later with aspect-oriented programming will enforce programmers to use standardized naming. In our project we have used JBOSS for developing aspect-oriented programming. We would also like to give a brief comparison between the aspectj and JBoss implementation. One of the most common aspect-oriented programming languages is aspectj whose documentation is actually better than the JBoss language. Therefore, not having enough experience on the subject make us use the documentation and web help for the problems encountered; however the documentation for JBoss is not adequate enough. Furthermore, the plug-in IDE for aspectj is quite better than the JBoss. The implementation of aspectJ is easier to understand compared to JBoss, however JBoss has advantages over aspectj on having the ability to provide dynamic AOP. What makes it harder to understand the implementation of JBoss is mostly the concept of "reflections" - the message passing procedure- in JBoss. However, once concepts are strengthened with aspectj, it becomes easier to understand. In aspectj, the codes are written in separate files with extension ".aj"; however JBoss codes are written into .java files, but only the pointcuts are written into a different file (".xml" files) and bind to the corresponding places with the keyword "bind".

7. CONCLUSION

In the implementation period of our e-commerce book store application, we have experienced the usage an advantages of aspect-oriented programming with the need of an improvement to the system. Even though the programmers will pay great attention in designing and implementing the projects in such a way that it will be helpful to make changes in the future, it might be quite difficult to foresee the upcoming technology and its requirements. Therefore, especially making changes to the system with aspect-oriented programming rather than object oriented programming is a high better way and makes the project for later use, more reusable and maintainable.

With AOP, adding authentication and authorization, new user types and even exception handling is much easier than the object oriented version of the solution.

On the other hand, it is worthwhile to mention that, using such new technology in programming language is a little bit difficult in the sense of finding documentation and samples on the web. However, once the basic idea is understood AOP becomes preferable over recently used methods.

Using aspect-oriented software development, helps programmers to write neater code, and enforces them to obey the code standards and design policies. This way, the programmer will not deal with the scattered and tangled code that results from crosscutting and handles the concerns without making changes all over the code.

8. ACKNOWLEDGEMENT

We would like to thank to Ass. Prof. Bedir Tekinerdogan, for teaching us the steps of the Aspect-oriented Software development during the CS 586 course and organizing the Third

Turkish Aspect-Oriented Software Development Workshop.

9. REFERENCES

[1] Lippert, M., Lopes C. V. A Study on Exception Detection and Handling Using Aspect-Oriented Programming

[2] Reina, A. M., Torres, Toro, M. Aspect-Oriented Web Development vs. Non Aspect-Oriented Web Development

[3] Kiryakov, Y., Galletly J., Aspect-Oriented Programming- Case Study Experiences, 2003

[4] Sun Microsystems Java Pet Store specification: http://developer.java.sun.com/developer/releases/petstore/

[5] Dounce, R., Motelet O., Südholt M., Sophisticated Crosscuts for E-Commerce,

[6] Dounce R., Südholt M., A Model and A Tool for Event-Based Aspect-Oriented Programming (EAOP), 2002

[7] JBOSS, http://www.jboss.org/

[8] http://www.ebuyer.com

[9] http://www.ebay.com/

[10] http://en.wikipedia.org/wiki/Electronic_commerce