ISSN No. 0976-5697

Volume 3, No. 1, Jan-Feb 2012

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

Use of Design Patterns in E-commerce Application:Survey

Pallavi S.Kulkarni*
Computer Technology Department
VJTI Mumbai, India
pskulkarni77@gmail.com

Pradnya B. Rane
Computer Technology Department
VJTI Mumbai, India
pradnyarane@gmail.com

Suchita P.Patil
Computer Technology Department
VJTI Mumbai, India
suchitapatil26@gmail.com

Dr.B.B.Meshram
Computer Technology Department
VJTI Mumbai, India
bbmeshram@vjti.org.in

Abstract: Design patterns are proven solution for common recurring problems. Design patterns which are described by Gang of Four has been used by developer for reuse, maintainability, flexibility and thus to improve quality of the software. As these are the abstraction of the ideas many variations are also given to the existing patterns to overcome some of the problems existing in the pattern implementation in some specific conditions as use of Singleton pattern in case of multithreading. This paper describes the patterns with applications of the patterns in the E-commerce applications.

Keywords: MVC, Observer, Memento, E-commerce, Template Method

I. INTRODUCTION

Design patterns are proven solution for common recurring problems like creating only one instance of the class in the application, maintaining states of the objects and delegation of the responsibilities according to state value etc.[1][2][3][4]
Patterns are mainly categories into four types
Creational Patterns: The patterns which are useful for creation of objects.
Behavioural Patterns: The patterns which are helpful for addressing the behavioural of the objects (functional interactions between objects.)
Structural Patterns: The patterns which are managing the structural relationship between objects.
System Patterns: The patterns which are useful for the system level interaction.[5]
The World Wide Web has become a popular platform for E-commerce applications.
These applications combine navigation through an electronic catalogue with operations affecting this catalogue. In this sense e-commerce applications are a particular kind of Web applications with similar requirements: good navigational structures, usable interfaces, a clear domain model, etc.[6]

However, E-commerce applications presently providing easy solution for online shopping. Here the user should find the product he is interested in and at the same time it should provide usability also. For example we should keep him informed about new releases and, keep him in the electronic shop for a longer time. As modern e-commerce applications become very complex, the need for lower maintenance costs becomes more and more obvious [6] It is likely that customers issue requests based on out-of-date information in e-commerce application systems. Some preference models has to be implemented in such situations. [7]

E-commerce applications are categories into different types

a) B2B – Business to Business E-commerce
b) B2C – Business to Consumer
c) C2C-Consumer to Consumer
d) B2E – Business to Employee
e) C2B-Consumer to Business
f) G2G- Government to Government

Use of Design patterns in E-commerce application is proven to be effective so as to reduce the development effort and improve the quality of the application.

At the same time developer has to keep in mind that design pattern can increase complexity sometimes in the development like if number of states are less in the application State Pattern will increase the complexity of the application. Here just simple If –then-Else will work.

This paper is organized as follows. Section I is introduction which gives brief ides about design patterns. Section II focused on the design patterns which are useful in the E-commerce Applications. Section III discusses the structure of the patterns. Paper concludes with section IV with application of patterns in E-commerce application.

II. RELATED WORK

Web applications are normally build from scratch. Design patterns are providing reusability in the web application design.[2] There are two types of patterns which are majorly used in the web application.

a. Interface Related Patterns – Patterns in this category are related to GUI and look and feel of the application.
b. Hypermedia Application Related Patterns –This category of patterns deal with the behavioural, creational or structural patterns for the web application like MVC, State pattern, Session Pattern etc.

All the patterns can be used with the application in different ways. Every pattern has some advantages and disadvantages.

III. DISCUSSION

This section discusses various design pattern used int E-commerce application in different ways.
Following patterns are useful in E-commerce application
Creational Patterns: Abstract Factory, Factory method, Prototype, Singleton
Structural Patterns: Decorator
Behavioral Patterns: Command, Observer, Iterator, State, Session, Transaction, CallBack, Template Method, Strategy, Memento
System Pattern: Model-View-Controller

A. Abstract Factory:

This is creational pattern which is used in the application where different objects to be created for different resources.
This pattern helps us to have a generic resource creator. The factory has one or more create methods, which can be called to produce generic resources or abstract products.[5][8]
This pattern has following structure

- ***a.*** ***Concrete Factory –*** A class derived from the abstract factory. It implements create methods for one or more concrete products.
- ***b.*** ***Concrete Product –*** A class derived from the abstract product, providing an implementation for a specific resource or operating environment.
- ***c.*** ***Variant:*** Multiple Concrete Factory can be produced which allows the application to use multiple families of Concrete Products.[5][8]

B. Factory Method:

Factory Method is also creational Pattern which is useful to create one type of objects. It is used to define a standard method to create an object, apart from a constructor. Subclass will decide what type of object is to be created.
The pattern has following structure

- ***a.*** ***Product –*** The interface of objects created by the factory.
- ***b.*** ***Concrete Product –*** The implementing class of Product. Objects of this class are created by the Concrete Creator.
- ***c.*** ***Creator –*** The interface that defines the factory method (s) (factory Method).
- ***d.*** ***Concrete Creator –*** The class that extends Creator and that provides an implementation for the factory Method. This can return any object that implements the Product interface.
- ***e.*** ***Variants:*** Creator can provide a standard implementation for the factory method. Product can be implemented as an abstract class. The factory method can take a parameter so that it can create more than one type of objects.[5][8]

C. Prototype:

It provides a copy method. That method returns an instance of the same class with the same values as the original Prototype instance. The new instance can be a deep or shallow copy of the original. [9][5]
The pattern has the following structure

- ***a.*** ***Prototype –*** Provides a copy method. That method returns an instance of the same class with the same values as the original Prototype instance. The new instance can be a deep or shallow copy of the original.
- ***b.*** ***Variants:*** Copy constructor –A copy constructor takes an instance of the same class as an argument and returns a new copy with the same values as the argument.[5]
- ***c.*** ***Clone method –***For the method to be usable on an instance, the class of that object has to implement the java. lang. Clonable interface to indicate that an instance of this class may be copied.[5]

D. Singleton:

This is also a creational pattern. Singleton is used when only one instance of the object is to be created. Some application requires a single object in the entire application. Singleton pattern allows the object to be created only once and then used by the entire application.
The pattern has the following structure

- ***a.*** ***Singleton –*** Provides a private constructor, maintains a private static reference to the single instance of this class, and provides a static accessor method to return a reference to the single instance.
- ***b.*** ***Variants:*** One variant can be having more than one copy. The benefit is that the rest of the application can remain the same, while those that are aware of these multiple instances can use other methods to get other instances.[5]

E. Command:

The command object decouples the object that invokes the action from the object that performs the action.

Three terms always associated with the command pattern are *client*, *invoker* and *receiver*. The *client* instantiates the command object and provides the information required to call the method at a later time. The *invoker* decides when the method should be called. The *receiver* is an instance of the class that contains the method's code.[5][8]
The pattern has following structure

- ***a.*** ***Command –*** The interface that defines the methods for the Invoker to use.
- ***b.*** ***Invoker –*** The invoker of the execute method of the Command object.
- ***c.*** ***Receiver –*** The target of the Command and the object that fulfills the request; it has all the information needed.
- ***d.*** ***Concrete Command –*** Implementation of the Command interface. It keeps a reference to the intended Receiver.

When the execute method is called, Concrete Command will call one or more methods on the Receiver.

Handling of call is done either with the approach that the class that implements the Command interface can just be a coupling between the invoker and the receiver, and forward all the calls directly. This makes the Concrete Command lightweight.

The Concrete Command can be the receiver and handle all the requests itself. This is most appropriate when there is no specific receiver for that request.

- ***e.*** ***Variants:*** Undo – The Command pattern lends itself to providing undo functions. To support an undo for only

the last command, the application needs to keep a reference only to the last command.[8]

Commands can be used several times in different contexts so they are to be placed in the history list.

F. Observer:

This is the behavioral pattern. It defines one to many dependencies so that when one object changes state all of its dependents are notified and updated automatically. When the subject changes the state then all the dependents are notified.[3][8][10] [9]

The pattern has the following structure

- ***a. Observable*** **–** The interface that defines how the observers/clients can interact with an Observable. These methods include adding and removing observers, and one or more notification methods to send information through the Observable to its clients.
- ***b. Concrete Observable*** **–** A class that provides implementations for each of the methods in the Observable interface. It needs to maintain a collection of Observers. The notification methods copy (or clone) the Observer list and iterate through the list, and call the specific listener methods on each Observer.
- ***c. Observer*** **–** The interface the Observer uses to communicate with the clients.
- ***d. Concrete Observer*** **–** Implements the Observable interface and determines in each implemented method how to respond to the message received from the Observable. Normally the application framework registers the specific observers to the Observable.
- ***e. Variants:*** Many to many relationship can be implemented in subjects and observer with Hash table or some other mapping mechanism. Observer can have some mechanism to notify the subjects about the deletion of the observer.

G. Iterator:

Iterator is normally used to travserse through collection like array, list,set etc. [9][5][8]

The Iterator described in Design Patterns provides the following fundamental operations:

a. First
b. Next
c. Is Done
d. Current Item

The pattern has following structure

- ***a. Iterator*** **–** This interface defines the standard iteration methods. At a minimum, the interface defines methods for navigation, retrieval and validation (first, next, has More Elements and get Current Item)
- ***b. Concrete Iterator*** **–** Classes that implement the Iterator. These classes reference the underlying collection. Normally, instances are created by the concrete Aggregate. Because of the tight coupling with the Concrete Aggregate, the Concrete Iterator often is an inner class of the Concrete Aggregate.
- ***c. Aggregate*** **–** This interface defines a factory method to produce the Iterator.
- ***d. Concrete Aggregate*** **–** This class implements the Aggregate, building a Concrete Iterator on demand. The Concrete Aggregate performs this task in addition to its fundamental responsibility of representing a collection of objects in a system. Concrete Aggregate creates the Concrete Iterator instance.
- ***e. Variants:*** Null iterators can be defined to make traversal of complex structures, such as trees, more straightforward. [5]

H. State:

Implementing this decision-making in the individual methods makes the code hard to maintain and read. The result is that these methods contain long if/ else statements. A common tactic is to store the state of an object in a single variable using constants for a value. With this approach the methods normally contain large switch/case statements that are very similar in each method. [9]

Objects are state and behavior; state is kept in its attributes and the behavior is defined in methods. The State pattern allows you to change the behavior of an object dynamically. This dynamic behavior is achieved by delegating all method calls that rely on certain values to a State object. Such a State object is state and behavior as well, so that when you change State objects, you also receive a different behavior. The methods in the specific State classes no longer have to use if/else or switch statements; the State object defines the behavior for one state.[5][8]

The pattern has following structure

- ***a. Context*** **–** Keeps a reference to the current state, and is the interface for other clients to use. It delegates all state-specific method calls to the current State object. State – Defines all the methods that depend on the state of the object.
- ***b. Concrete State*****–** Implements the State interface, and implements specific behavior for one state.

The Context or the Concrete State can determine the transition between states. This is not specified by the State pattern. When the number of states is fixed, the most appropriate place to put the transition logic is in the Context.

Variants: Enumerated type or hash table can be used to save the values of states. When the states are less then just have a conditional statement usage.

I. Session:

Session is used to provide a way for servers in distributed systems to distinguish among clients, allowing applications to associate state with the client-server communication. Session specific data has to be stored in between requests, and made available to the code handling a request either on client or on server. Session pattern can be used as state ful and stateless. States can be stored at server or at client.[5]

The pattern has following structure

- ***a. Session Tracker:*** It will create and destroy sessions.
- ***b. Session:*** It will have session Id.
- ***c. Variants:*** Variants can be saving the data related to session on server or client. If it is saved on the server assign a token to the individual session. Use this token as a key into the data structure in the server that holds the session specific data for all clients. If the data is saved on the client then the data has to be transferred to the server every time when the request is made.

J. Decorator:

The Decorator Pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. It provides a way to flexibly add or remove component functionality without changing its external appearance or function.[1]

The pattern has following structure

- ***a. Component*** **–** Represents the component containing generic behavior. It can be an abstract class or an interface.
- ***b. Decorator*** **–** Decorator defines the standard behaviors expected of all Decorators. Decorator can be an abstract class or an interface. The Decorator provides support for containment; that is, it holds a reference to a Component, which can be a Concrete Component or another Decorator. By defining the Decorator class hierarchy as a subclass of the component(s) they extend, the same reference can be used for either purpose.
- ***c. One or more Concrete Decorators*** **–** Each Decorator subclass needs to support chaining (reference to a component, plus the ability to add and remove that reference). Beyond the base requirement, each Decorator can define additional methods and/or variables to extend the component.[5]
- ***d. Variants:*** Decorator pattern can be used with XML files for creating different GUIs in case of web application.[1]

K. Model View Controller:

It is possible to view the component or subsystem in different ways. The internal representation for the system may be completely different from the representation on a screen.[6][9][11]

There are different possible types of behavior, meaning that multiple sources are allowed to invoke behavior on the same component but the behavior may be different. Behavior or representation that changes as the component is used.[8] [12]

- ***a. Variants:*** View resolver can be added to render the proper view to the user. View resolver can be a Decorator pattern which helps if to generate different GUI's depending on the requirement or role of user. Many of the frameworks are having XML file as a configuration file for selection of the view.

L. Transaction:

To group a collection of methods so that they either all succeed or they all fail collectively.

If we are transferring funds from one account to another account then if only withdrawal from one account takes place and another account is not deposited then it is a wrong transaction so here both the events should happen or both should fail.

Transaction makes sure that the transaction is either successful completely or it is failed completely.

The pattern has the following structure

- ***a. Transaction Participant–*** The interface that defines the methods to control every participant.
- ***b. Specific Participant*** **–** As an extension to the generic interface, this interface contains the business methods. All methods involved in the transaction take an ID as parameter. Methods involved in transaction can throw Exceptions as a signal of failure.
- ***c. Concrete Participant–*** Implements Specific Participant interface. It defines what happens if the transaction manager (in this case, client) decides to roll-back or commit. It has to keep a reference to the original state to be able to restore it when cancel is invoked.
- ***d. Client*** **–** Acts as transaction manager. The client calls the join method on the participants to start the transaction and ultimately calls either cancel or commit on the participants.[5]
- ***e. Variants:*** Two-phase commit is the variant of transaction where the manager asks the sub ordinates about the transaction status.

M. Call Back :

This pattern is behavioural pattern which is used to allow a client to register with a server for extended operations. This enables the server to notify the client when the operation has been completed.

Use the Callback pattern for a client-server system with time-consuming client operations.

The Callback pattern provides this capability, allowing for asynchronous client-server communication. The process consists of three simple steps Client registration, Server processing, Server call back[5]

The pattern has following structure

- ***a. Sender:*** This will send the request.
- ***b. Receiver:*** Who will receive the request
- ***c. Variants:*** Queuing of Client request can be possible with the help of command object and then the queue will be resolved with the help of multithreading.[5]

N. Template Method :

This is a behavioural Pattern. This pattern is used to provide a method that allows subclasses to override parts of the method without rewriting it. [9][5][8]

Template Method is giving the template for the method which can be overridden for the customized behaviours.

The Pattern has the following structure

- ***a. Abstract Class***– The Abstract Class is (perhaps not surprisingly) an abstract class that contains the template method and defines one or more abstract methods. The template method contains the skeletal code and calls one or more of the abstract methods. To prevent subclasses from overriding the template method it should be declared final.
- ***b. Concrete Class*** – The Concrete Class extends the Abstract Class and implements the abstract methods of the Abstract Class. It relies on the Abstract Class to provide the structure of the operation contained in the template method.
- ***c. Variants:*** One variant is to have the Template Method call concrete methods instead of abstract methods. These methods are called hook methods.[8]

O. Memento:

This can be used for producing an object to maintain a snapshot of state that cannot be modified by other objects in a system.

Delegate some activity to some other class like some helper classes to do the job.[5]

The memento pattern is used to encapsulate the current state of an object in a memento object in order to be able to restore the object state later without exposing the internal representation of the object to the outside world.

The pattern has the following structure

- ***a. Originator –*** Creates Memento and uses this Memento to later restore its state.
- ***b. Memento –*** Static inner class of the Originator and holder of the Originator's state. The Originator determines how much is stored in the Memento and only the Originator should be able to read the Memento. State within the Memento should be inaccessible to everybody except the Originator.
- ***c. State Holder –*** The object that wishes to preserve the state. It never needs to know what is within a Memento; it only needs to know that the object it receives enables it to restore the state of the Originator.
- ***d. Variant:*** To be able to extend the Originator but not need to change the Memento code, the methods can have package access instead of being private. This allows subclasses of the Originator to use the same Memento class.[5]

P. Strategy Pattern:

This is a behavioral pattern. It is used to define a group of classes that represent a set of possible behaviors. These behaviors can then be flexibly plugged into an application, changing the functionality on the fly. .[5][8][9][10]

The pattern has the following structure

- ***a. Strategy Client*** – This is the class that uses the different strategies for certain tasks. It keeps a reference to the Strategy instance that it uses and has a method to replace the current Strategy instance with another Strategy implementation.
- ***b. Strategy –*** The interface that defines all the methods available for the Strategy Client to use.
- ***c. Concrete Strategy*** – A class that implements the Strategy interface using a specific set of rules for each of the methods in the interface.[5][9]
- ***d. Variants:*** Strategy used with Decorator pattern for changing the entire functionality of the object.

IV. CONCLUSION

Many of the existing design patterns are used in the E-commerce application in association with other patterns. Mainly MVC pattern is used in E-commerce application. Pattern has some draw backs. Some variations can be added to patterns to make it efficient. Design patterns are proven solution to re-occurring patterns.

Design patterns are proven design building blocks that benefit the design process by (1) reusing design early in the development lifecycle, (2) reducing development effort and cost, (3) increasing software quality, and (4) providing a common vocabulary for design among different stakeholders.[13]

Some patterns are provide hints to the Web application designer in order to make these applications more usable and effective both from the point of customers and owners of the store like push communication, advising etc.[14]

Table I List of Patterns with use in E-commerce Application

Pattern Name	Type	Use in E-commerce
Abstract Factory	Creational Pattern	This pattern is useful in E-commerce application when the shipping address is to be stored. As the addresses are different in different countries this pattern is useful to create address and phone number as two countries can have different format for the address and phone number field.[5]
Factory Method	Creational Pattern	Lots of ecommerce applications, provide profile management by storing static and dynamic information about users. Providing individual or multiple users access to resources such as individual files, directory structures, user-defined entities (such as catalogs, purchase orders etc.) etc., can be controlled based on the individual/group's organization/role with and within an organization.[5]
Prototype Pattern	Creational Pattern	When a user asks for information for a certain product the application could gather that information by instantiating the objects at predefined intervals and keep them in a cache, when an object is requested, it is retrieved from cache and cloned. When the legacy database is updated, discard the content of the cache and re-load with new objects.[8]

Singleton Pattern	Creational	History List is the major application of Singleton. Another application is the Database Connection. For the entire application only one connection object is created and then that object is used.
Command Pattern	Behavioural Pattern	When history list is created is grows as the application use is increased. Here the command object can be used as a history list. Just add the commands in the object and then execute the commands through execute command.[8]
Observer Pattern	Behavioural Pattern	When the item is not in the inventory observer pattern can be useful to notify user about the availability of the item. Second use is notificatio to the user in case of shipping the item. The item undergoes various stages like shipped delivered such phases can be notified to user through Observer Pattern.
Iterator pattern	Behavioural Pattern	Iterator pattern is normally used for the traversal through the item list in the application.
State Pattern	Behaviroal Pattern	E-commerce Application can use state pattern to handle the states of use or the system. The user in expects to see the following state transitions when using the ecommerce program: the initial state, Moving from the initial state into the purchase state,Moving from the purchase state into the exit state,Moving from the exit state to the end of the program or back to the initial state.
Session Pattern	System Pattern	To maintain the state of the client as a session so that the validations or purchase can be associated with the session pattern. Series of business transactions is maintained with the use of the Session Pattern.
Decorator Pattern	Structural Pattern	While creating or rendering GUI, decorator helps us to create different GUIs depending on the user role. Decorator pattern can be used with XML file to do this. It reduces extension to sub classing.
Transaction Pattern	System Pattern	In E-commerce Application in payment transaction and in Shopping cart application Transaction pattern can be used.
CallBack Pattern	Behavioural Pattern	When user selected the notify option with observer pattern the subject means the mail object will be revoked and then the mail will be sent to the user about the availability of the Item.
Template Method	Behavioural Pattern	In case of E-commerce Application, while executing query, template method is used. With this pattern, the query interface is created .This can be used with different databases by changing the algorithm steps for connection and selection.
Memento Pattern	Behavioural Pattern	Memento pattern is used in case of Undo operation where the previous state of the originator is stored.[8]
Strategy Pattern	Bahavioural Pattern	In E-commerce application while the time of discount calculation strategy pattern is used.
Model View Controller	System Pattern	Model View Controller separates the logic into user interface, business layer and database layer.E-commerce application can use MVC to make the code maintainable and to improve the quality of the same.[8]

V. REFERENCES

[1] Steve Macdonald,Kai Tan,Jonathan Schaeffer amd Duane Szafron, "Deferring Design Pattern Decisions and Automating Structural Pattern Changes Using a Design-Pattern-Based Programming System",ACM Transactions on Programming Languages and Systems, Vol. 31, No. 3, Article 9, Pub. date: April 2009, DOI 10.1145/1498926.1498927 http://doi.acm.org/10.1145/1498926.1498927,1-48

[2] Allan Shalloway,James R.Trotta, ," The Principles and Strategies of Design Patterns", Net Objectives EZINE volume 1 Issue 4,April 2004,pp:1-13

[3] Giuseppe Antonio Di Lucca, Anna Rita Fasolino, Porfirio Tramontana, ,"Recovering Interaction Design Patterns in Web Applications", Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR'05) 1534-5351/05 © 2005 IEEE,pp:not available

[4] Jeffrey Heer and Maneesh Agrawala, "Software Design Patterns for Information Visualization", IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 12, NO. 5, SEPTEMBER/OCTOBER 2006,pp:853-860

[5] Stephen Stelting,Olav Maassen,"Applied Java™ Patterns", ,Publisher: Prentice Hall PTR First Edition December 01, 2001,pp:12-31,41-45,52-56,63-85,93-96,114-119,134-138,140-154,160-166,178-182

[6] Maria Mouratidou, Vassilios Lourdas, Alexander Chatzigeorgiou and Christos K. Georgiadis, "An Assessment of Design Patterns' Influenceon a Java-based E-Commerce Application", Journal of Theoretical and Applied Electronic Commerce Research ISSN 0718–1876 Electronic Version VOL 5 / ISSUE 1 / APRIL 2010 / 25-38 © 2010 Universidad de Talca – Chile, DOI: 10.4067/S0718-18762010000100004,pp:25-38

[7] Peng Li ,Manghui Tu,I-Ling Yen, Zhonghang Xia**,** **"**Preference update for e-commerce applications: Model,language, and processing ",Springer, Electron Commerce Res (2007) 7:17–44 , Springer Science+Business Media, LLC 2007,DOI 10.1007,pp:17-44

[8] Eric freeman and Elisabeth Freeman ,"Head first design patterns",Orielly publication,1st Edition, October 2004,pp:1-191,275-300,385-495

[9] Bruce Eckel ,"Thinking in Patterns",Revision 0.9, 5-20-2003,pp:8-75

[10] Phek Lan Thung, Chu Jian Ng, Swee Jing Thung, Shahida Sulaiman,"Improving a Web Application Using Design Patterns: A Case Study", 978-1-4244-6716-7/10/$26.00 ©2010 IEEE,pp:not available

[11] Patrick Sauter ,Gabriel Vogler, Gunther Specht Thomas Flor,"A Model–View–Controller extension for pervasive multi-client user interfaces", Pers Ubiquit Comput (2005) 9: 100–107 Springer-Verlag London Limited 2004, 1 October 2004, DOI 10.1007/s00779-004-0314-7,pp:100-107

[12] Avraham Leff, James T. Rayfield ,"Web-Application Development Using the ModelNiewlController Design Pattern",0-7695-1345-2UOl, 2001 IEEE,pp:118-127

[13] Jaeyong Park,David C. Rine,Elizabeth White, "Assessing Conformance of Pattern-based Design in UML", ACM-SE '08, March 28–29, 2008, Auburn, AL, USA. Copyright 2008 ACM ISBN 978-1-60558-105-7/08/03,pp:298-303

[14] Gustavo Rossi, Fernando Lyardet, Daniel Schwabe," Patterns for E-commerce applications", In Proceedings of Europlop 2000, pp: 1-13, doi=10.1.1.31.6668