E-COMMERCE PATTERNS

By
Li Yan

A thesis submitted to the
School of Computer Science
and the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

April 2004

©Copyright 2004, Li Yan

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-97460-X

Our file Notre référence

ISBN: 0-612-97460-X

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

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives 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.

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 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.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

PAGINATION ERROR.

ERREUR DE PAGINATION.

TEXT COMPLETE.

LE TEXTE EST COMPLET.

ABSTRACT

Software patterns establish knowledge for a solution to a programming problem in a context. With the increasing adoption of patterns, a large number of patterns are now available to designers or pattern users. Under these circumstances, pattern users are challenged to determine which pattern to incorporate into their designs.

This thesis presents an index and a new classification of e-commerce patterns, surveys existing e-commerce patterns and rewrites them in a consistent format, and uses an extended pattern representation based on force hierarchies. The use of the resulting pattern system is illustrated with a case study.

ACKNOWLEDGEMENT

First of all, I would like to express my sincere thanks to my thesis supervisor, Dr. Michael Weiss, for his generous support, profound guidance and continual encouragement during my research. His valuable experience and his attitude toward research guided me through this thesis, and will guide my future study and work.

Deep appreciation is expressed to Donald D Wachenschwanz and Tao Wan, for their role in reviewing this thesis during final preparations. Their observations were very helpful in enhancing the readability and understanding of the thesis content.

With utmost gratitude this thesis is dedicated to my family. Without their support, their encouragement and love, this journey would not have been feasible.

Contents

Abstract	iii
Acknowledgements	iv
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Goal	2
1.3 Contributions	2
1.4 Thesis Structure	4
2 Background	6
2.1 Concepts in Software Architecture	6
2.2 Non-Functional Requirements and Patterns	7
2.3 Chapter Summary	8
3 NFR Framework and Related Work	9
3.1 The NFR Framework	9
3.2 Force Hierarchy	12
3.3 Chapter Summary	13
4 Patterns and Catalogs	14
4.1 Patterns and Pattern Languages	14
4.1.1 Patterns	14
4.1.2 Pattern Languages	15
4.2 Representing a Pattern	16
4.2.1 The Alexandrian Form	16
4.2.2 Structure of Software Design Patterns	17
4.2.3 Rewriting Patterns Using Forces Hierarchies	19
4.3 Application Domain Forces	20

4.4 Pattern Solution Force Hierarchy.....	23
4.5 Roles in Pattern Application	25
4.6 Classification of Patterns	27
4.6.1 Granularity	27
4.6.2 Functionality	28
4.6.3 Structural Principles.....	29
4.6.4 Comparison.....	30
4.6.5 Categories of E-commerce Patterns.....	30
4.7 Chapter Summary	36
5 E-commerce Patterns.....	37
5.1 E-commerce Pattern Roadmap	37
5.2 UI Patterns	40
5.2.1 Security Patterns	40
5.2.2 Page Content Patterns	68
5.2.3 Page Elements Patterns	86
5.3 Navigational Patterns.....	103
5.3.1 Process indication Patterns	103
5.3.2 Communication Patterns.....	111
5.3.3 Search Patterns.....	134
5.4 Infrastructure Patterns.....	157
5.4.1 Process Patterns	157
5.4.2 Atomic Patterns.....	179
5.5 Server-side Patterns	183
5.6 Chapter Summary	205
6 Case Study: A Main Feature of A Web Store	206
6.1 Selecting the Patterns.....	207
6.2 Analyzing the Issues in Purchase Process	209
6.3 Chapter Summary	212

7 Conclusions.....	213
7.1 Review of the Objectives.....	213
7.2 Summary of the Contributions.....	214
7.3 Future Work.....	215
Appendix	210
Bibliography	220

LIST OF TABLES

Table 4.1 Forces and Non-functional features in e-commerce system	21
Table 4.2 Resources involved in an e-commerce system design context	22
Table 4.3 The index of patterns for e-commerce application	32
Table 5.1 Force Table for Security Patterns	41
Table 5.2 Force Table for Page Content Patterns	68
Table 5.3 Force Table for Page Elements Patterns	86
Table 5.4 Force Table for Process indication patterns.....	103
Table 5.5 Force Table for Communication Patterns	111
Table 5.6 Force Table for Search Patterns.....	135
Table 5.7 Force Table for Process Patterns	158
Table 5.8 Force Table for Atomic Patterns.....	179
Table 5.9 Force Table for Server-side Patterns.....	184

LIST OF FIGURES

Figure 3.1	NFR framework example of paying and locating the teller(s)	10
Figure 3.2	Distinction among contribution types	11
Figure 4.1	Force hierarchy for e-commerce system design	24
Figure 4.2	E-commerce stakeholder interactions	26
Figure 4.3	N-tired logic model	31
Figure 5.1	Interrelationship of e-commerce patterns	39
Figure 5.2	Force hierarchy of Login, Single Access Point, Check Point, Policy Statement and Roles	50
Figure 5.3	Example of website using <i>Footer Bar</i> pattern and <i>Premium Content Lock</i>	56
Figure 5.4	Force hierarchy of Patterns in registration processes	67
Figure 5.5	Force hierarchy for the <i>Language Selector</i> pattern	82
Figure 5.6	Force hierarchy for Page Controller, Front Controller and Central Dispatcher	84
Figure 5.7	Force hierarchy for the <i>Template View</i> pattern and the <i>Transform View</i> pattern	85
Figure 5.8	Example of website using the <i>FAQ</i> pattern	89
Figure 5.9	Example of website using the <i>List Builder</i> pattern	92
Figure 5.10	Force hierarchy for the <i>Product Configurator</i> pattern	97
Figure 5.11	Force hierarchy for the <i>Product Comparison</i> pattern	99
Figure 5.12	Force hierarchy for page element patterns	102
Figure 5.13	Example of website using the <i>Explicit Process</i> pattern and the <i>Easy Undo</i> pattern	109
Figure 5.14	Force hierarchy for <i>Easy Undo</i> , <i>Explicit Process</i> and <i>Process Feed-back</i>	110
Figure 5.15	Example of website using the <i>Opportunistic Linking</i> pattern	113
Figure 5.16	Force hierarchy for the <i>News</i> pattern	116

Figure 5.17 Example of website using the <i>News</i> pattern.....	117
Figure 5.18 Example of website using the <i>Hotlist</i> pattern.....	122
Figure 5.19 Force hierarchy for the <i>Dynamic Configuration</i> pattern	126
Figure 5.20 Example of website using the <i>Dynamic Configuration</i> Pattern	127
Figure 5.21 Example of website using <i>My Site</i> pattern	129
Figure 5.22 Stucture-less structure	131
Figure 5.23 Star-shaped structure	131
Figure 5.24 Force hierarchy for patterns in communication.....	133
Figure 5.25 Force hierarchy for the <i>Simple Search Interface</i> pattern.....	137
Figure 5.26 Force hierarchy for the <i>Selectable Keywords</i> pattern.....	139
Figure 5.27 Force hierarchy for the <i>Structured Answer</i> pattern	141
Figure 5.28 Example of website using the <i>Structured Answer</i> pattern.....	142
Figure 5.29 Force hierarchy for the <i>Advanced Search</i> pattern	144
Figure 5.30 Force hierarchy for the <i>Selectable Search Space</i> pattern	146
Figure 5.31: Example of website using the <i>Selectable Search Space</i> pattern	147
Figure 5.32: Force hierarchy for the <i>Search Tips</i> pattern	149
Figure 5.33: Force hierarchy for the <i>Selectable Search Engine</i> pattern	151
Figure 5.34 Force hierarchy for the <i>Search Area</i> pattern	154
Figure 5.35 Force hierarchy for several search patterns	156
Figure 5.36 Force hierarchy for the <i>Catalog</i> pattern	160
Figure 5.37 Force hierarchy for the <i>Shopping Basket</i> pattern	163
Figure 5.38 Example of website using <i>Shopping Baske/Cart</i> pattern	164
Figure 5.39 Force hierarchy for the <i>Transition Table</i> pattern	166
Figure 5.40 Force hierarchy for the <i>Continuation</i> pattern	168
Figure 5.41 Force hierarchy for the <i>Order/Shipment</i> pattern	172
Figure 5.42 Force hierarchy for the <i>Shopping Process</i> pattern	175
Figure 5.43 Force hierarchy for the <i>E-commerce Site</i> pattern	178
Figure 5.44 Force hierarchy for the <i>Value Object</i> pattern	181

Figure 5.45 Force hierarchy for the <i>Unique Id t</i> pattern	188
Figure 5.46 Force hierarchy for the <i>Client Session State</i> pattern	190
Figure 5.47 Force hierarchy for the <i>Server Session State</i> pattern.....	192
Figure 5.48 Force hierarchy for the <i>Database Session State</i> pattern.....	197
Figure 5.49 Force hierarchy for the <i>Serialized LOB</i> pattern.....	199
Figure 5.50 Force hierarchy for patterns on sessions	201
Figure 5.51 Force hierarchy for the <i>Application Server</i> pattern	203
Figure 5.52 Sample of Application Server using in J2EE	208
Figure 6.1 Seleting patterns using pattern roadmap.....	211
Figure 6.2 Force hierarchy for some patterns of e-store.....	192

1 Introduction

1.1 Motivation

With the increasing adoption of patterns in software design and the number of extension to existing patterns documented as new patterns, a large number and great diversity of patterns are now available to the designer. A number of international conferences on software design patterns are held yearly, such as PLoP [PLoP], EuroPLoP [EuroPLoP] and KoalaPLoP [KoalaPLoP]. Hence, it is no longer practical to scan and choose from among a list of pattern titles as originally suggested [Alexa77]. Linda Rising writes in the introduction to her book on design patterns in communication software [Rising00B] that the troubling aspect with the large number of patterns available is that the pattern community feels overwhelmed by it. She is also the author of The Pattern Almanac 2000 [Rising00A] in which she compiled the patterns that had been published by that date. Under these circumstances, the designer or pattern user is faced with a dilemma as to which pattern to choose to incorporate into their design.

To drive the whole system design process, Chung proposed the *non-functional requirement (NFR) framework* [Chung00] focusing on non-functional quality attributes, such as security, and efficiency. By means of NFR, Araujo presents a new pattern representation using the notion of force hierarchy [Araujo02A] to help pattern users easily grasp the idea behind patterns, which is adapted in this thesis. In this thesis, a pattern index and a new classification of e-commerce patterns, as well as their interrelationships are provided. Documenting the connections between patterns is the basis for the creation of a pattern language. Thus, our goal is to help pattern users choose appropriate patterns with support of our new pattern format, pattern classification and pattern map in the design process. The following section will outline our thesis goal.

1.2 Thesis Goal

As the previously documented patterns in separate papers use different formats, no consistent set of forces and links to patterns in other pattern languages has been identified. The goal of this thesis is to present these patterns in a consistent style as well as to represent patterns and trade-offs between them through a new representation based on the NFR framework. Furthermore, classifying related patterns into categories will facilitate the selection of patterns. As a result of our pattern classification, an overview of the interrelationships of e-commerce patterns selected in this thesis will be provided in the form of a pattern roadmap.

1.3 Contributions

The expected contributions of the thesis are as follows:

- It will provide an index of patterns for e-commerce applications. Our index of patterns in e-commerce application aids pattern users in easily finding which pattern they might need in the early stage of system design, according to the brief description, pattern type and page reference.
- It will provide a new classification of e-commerce patterns. There exist various pattern classifications since the original pattern authors have classified their patterns in their own manner. It is not easy for pattern users to make a choice.

Thus, the patterns collected in this thesis are classified into four categories offering support for selecting and applying patterns.

- It will present the patterns in a consistent manner. A typical pattern may contain several names and be represented in different ways. Our consistent pattern representation refines the pattern description by explicitly expressing forces, which provides supports for understanding contributions and trade-offs. In the original patterns, sometimes the pattern forces are already explicit, but in a format that makes it hard to compare with other patterns. Sometimes different descriptions are given of the same force. Hence, we establish a common set of forces in e-commerce domains and replace the existing forces in the individual patterns, which is supplied in chapter 5.
- It will document previously undocumented links between e-commerce patterns. The interrelationships of the compiled patterns are typically not documented, therefore we provide the interrelationships by adding more links between them in the pattern map.
- It will provide a pattern map on the interrelationships of the compiled patterns. With the support of new pattern representations and considering the way patterns are interrelated, a more complete picture of pattern interrelationships may be derived as well as several new pattern languages.
- It will provide a new approach to compare different patterns using force hierarchies. We place several related patterns in one force hierarchy. Through comparing and analyzing the non-functional requirements and forces, the pattern users will easily identify the contributions and side effects on the whole system when using this set of patterns. Hence, it may help to obtain a pattern-oriented architecture in accordance with pattern comparison in system development.

1.4 Thesis Structure

The rest of the thesis is organized into five parts:

- 1 **Background:** This chapter initiates this work by giving the concepts of software architecture design, and then provides an overview of associated material relevant to the conceptual framework of our proposed approach, such as non-functional requirement and patterns.
- 2 **NFR Framework:** The NFR framework uses non-functional requirements, which are presented as soft-goals, such as interoperability, security, and performance to manage the overall design process. It serves as foundation for our work. Chapter 3 will explore how patterns and non-functional requirements may be linked through the use of the NFR framework in more detail.
- 3 **Patterns Format and Pattern Catalogs:** Based on an extended Alexandrian pattern form, another NFR-based pattern representation using force hierarchies is introduced. This essential pattern representation is employed to analyze an individual pattern in the chapter 4. Through analyzing the drawbacks and benefits, we derive our new pattern catalogs in terms of the n-tire architecture of web application.
- 4 **E-commerce Patterns:** This chapter compiles published patterns for e-commerce applications. These patterns are cataloged into four categories: user interface patterns, navigational patterns, infrastructure patterns and server-side patterns. A set of related patterns under the same context, such as for a user interface, is grouped in one table with the common forces. This table serves to help identify the negative and positive supports for the target soft-goals. Each pattern is addressed by our new pattern representation, which explicitly lists forces and analyzes the consequence with force hierarchy.
- 5 **Case Study:** This chapter serves to illustrate how the new pattern representation and pattern roadmap help designers know how and when to apply appropriate patterns

during the design process using a case study in which the pattern language is used to analyze and design the main features of an online store.

- 6 **Conclusion:** This part summarizes the manner in which the thesis meets the stated objectives and confirms the claimed contributions. The thesis concludes with a statement of suggested areas for future research.

2 Background

2.1 Concepts in Software Architecture

Software architecture is concerned with how to design software components and make them work together, which is defined as follows [Buschmann96]:

Software architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system.

A **component** is an encapsulated part of a software system. A component may have an interface. Components serve as the building blocks for the structure of a system. At a programming-language level, components may be represented as modules, classes, objects or a set of related functions [Buschmann96].

If we develop a software system for a customer, the first step is meeting with the customer to determine the requirements. A **requirement** is a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose [Pfleeger87]. To help describe requirements, Yeh and other experts divide requirements into two categories: functional and non-functional requirements [Yeh82]. The functional and non-functional properties are also expressed as functional and non-functional requirements.

The **functional requirement** is defined as follows in [Thayer90]:

A system/software requirement that specifies a function that a system/software system or system/software component must be capable of performing. Functional requirements are software requirements that define behaviors of the system, that is, the

fundamental processes of transformations that software and hardware components of the system perform on inputs to produce outputs.

Functional requirements describe the behaviors (functions or services) of the system in supporting some tasks or activities. The components are used to implement the functional requirements.

The **Non-functional requirements**, including constraints and qualities are always tied up with functional requirements [King96], and are defined as follows in [Thayer90] :

Software requirements that describe not what the software will do, but how the software will do it; for example, software performance requirements, software external interface requirements, software design constraints, and software quality attributes.

Expressed in natural language, functional requirements refer to a set of functions the system offers, while non-functional requirements refer to the manner in which such functions are performed. For example, for the functional requirement “print a file”, a non-functional requirement may be print the file “quickly”. Non-functional requirements define properties of a system such as performance, efficiency, reliability, and reusability.

2.2 Non-Functional Requirements and Patterns

Requirements analysis is one of the most crucial stages in the software design process. During this phase, much of the quality aspects are taken into account by the system designer. System qualities are often presented as non-functional requirements, such as usability, scalability, maintainability, performance, space and cost, which are critical to system success. The idea of design patterns is to offer a body of empirical

design information [Coplien96] that has proven itself, and that can be used during new design efforts.

Gross and Yu [Gross01] propose a systematic approach to organizing, analyzing and refining non-functional requirements by explicitly representing non-functional requirements in the descriptions of patterns. The strategy explicitly representing non-functional requirement in design patterns provides much support for structuring, understanding and applying design patterns during design. This aids pattern users in better understanding the rationale of a design, and makes patterns more amenable to structuring and analysis.

They extract non-functional requirements as soft-goals, and explain their contributions to each other in the problem section of each pattern. A pattern is presented by a soft-goal hierarchy in which solutions are represented as operationalizations, since they turn goals into solutions. The goal graph shows how well the pattern achieves the stated non-functional softgoals.

2.3 Chapter Summary

In this chapter, we reviewed some relevant concepts to the thesis including: *software architecture, component, requirements, functional requirements and non-functional requirements*.

Requirements analysis is one of the most crucial stages in the software design process. The approach of linking patterns and non-functional requirements through the use of NFR framework is the basis for the pattern representation used in this work. The next chapter will further explore the use of the NFR framework in our pattern representation.

3 NFR Framework and Related Work

The use of NFR framework to represent patterns is one of the core concepts of this thesis. We will review several aspects of the NFR framework relevant to the formulation and its usage in force hierarchies.

3.1 The NFR Framework

The NFR framework uses non-functional requirements such as performance, scalability, and security, to express overall design issues and help designers make decisions during the design process. Non-functional requirements are represented as soft-goals [Chung00]. With the aid of analyzing and decomposing soft-goals, the solution of implementation might be devised. The entire representation of soft-goals plus their decomposition is defined as a soft-goal graph [Chung00]. A soft-goal graph consists of one or more soft-goals. Each soft-goal can be decomposed into one or more sub-goals, which refines a “parent” soft-goal or interdependency links.

The Figure 3.1 illustrates the notation of a soft-goal graph through an example. The thin-line clouds represent soft-goals and sub-goals, and lines connecting soft-goals denote interdependency links. Claims justify the choice of contribution type on a link, which are denoted as dotted-line clouds. Correlation links indicate side effects on the system, and are represented as dotted-line links. Each link has a label with a degree for supporting target soft-goal, for example “break”, which means the selected solution will

break meeting the soft-goal at the end of link. The degree for supporting the parent soft-goal of a link becomes stronger from left to right side, which is shown in the Figure 3.2.

Figure3.1 : NFR framework example of paying and locating the teller(s) (adapted from [Araujo02])

The solid links between soft-goals indicate contributions between them, the types of which are: Break, Hurt, Some-, Help, Make, and Some+. They are defined by the notation in Figure 3.2 as follows [Chung00]:

Figure 3.2 : Distinction among contribution types. (adapted from [Araujo02])

Break means the child at start of the link will break the meeting of the parent soft-goal at the end of the link; *Hurt* shows partial negative support to the parent goal; and *some--* indicates negative contribution, either *Break* or *Hurt*.

By contraries, *Make* means the offspring sufficiently support the parent goals at the end of the link; *Help* indicates a positive but not sufficient contribution to the parent goals; *Some+* means there is some positive contribution, either *Help* or *Make*.

And means all sub-goals must be achieved to accomplish the parent goal. *Or* indicates only any of them is sufficient for satisfying the parent goal. They are used to assistant propagate the degree of achieving soft-goals through the contribution links.

Labels are used to determine the impact on higher-level soft-goals during the NFR framework evaluation process. This process is initiated by assigning labels indicating whether a pattern is selected or not. There are five type of labels: Denied, Weakly Denied, Undecided, Weakly Satisfied, and Satisfied.

As described in [Araujo02], all the forces clouds are placed at the top level following with constraints clouds at the next layers in the graph. Once a soft-goal is sufficiently refined, patterns are derived, which is represented by thick-line clouds at the bottom. At the same time, the trade-offs showing the conflict between each constraint

come out. The different operationalizations may indicate the different possibilities to reach the soft-goals, through which the designers may choose the specific solutions for the target system by analyzing the trade-offs.

3.2 Force Hierarchy

As described in [McPhail00], “total satisfaction” of design decision is influenced by the value of a number of decision makers. It is called software quality. Whereas the traditional approach to making design decisions in software engineering is based on this quality. McPhail also points out that traditional software engineering structures and views on quality have been reviewed to draw a representative value tree to be applied to pattern decision analysis. This value tree with the relationships between forces and a distinction among different types of force established is called a “force hierarchy”.

Another definition of force hierarchy defined in [Chung00] is that a force hierarchy is a hierarchical graph of forces and their contributions on each other, which is inspired by NFR soft-goal hierarchy.

According to [Araujo02A], forces are described as a combination of a small set of constraints on common resources: time, space, money, and effort. Thus, the trade-offs between non-functional requirements can be mapped to trade-offs between lower-level associated resources.

In consideration of previous experience, the force hierarchy provides a promising support for pattern decision analysis distinguishing between different levels of forces. An overall force hierarchy for the e-commerce domain is shown in Figure 4.1 in the next chapter. It shows patterns of e-commerce domain and incorporates the contributions of each pattern to the involved resources. Force hierarchy is used to analysis the result context of every pattern and to compare with other patterns under the same context in the chapter 5.

3.3 Chapter Summary

In this chapter we have introduced several concepts, such as soft-goal graphs, forces, constraints, contributions, and force hierarchy, when applying the NFR framework.

We have also reviewed the notation used in soft-goal graphs for developing a force hierarchy that establishes a clear link between non-functional requirements and functional elements in pattern oriented design analysis.

4 Patterns and Catalogs

4.1 Patterns and Pattern Languages

Although a single pattern provides a solution for a particular design problem, in reality patterns cannot be used on their own. It is like that when you build something you cannot merely build that thing in isolation [Alexa77]. Therefore, each pattern can exist in the world only to the extent that is supported by other patterns. Together such a set of patterns forms a pattern language.

4.1.1 Patterns

Christopher Alexander defines the term pattern in [Alexa97] as follows:

The pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when we must create it. It is both a process and a thing; both a description of a thing which is alive, and a description of the process which will generate that thing.

The goal of patterns within the software community is to create a body of literature and help software developers resolve recurring problems encountered throughout the whole development lifecycle. Patterns help create a shared language for communicating insight and for recording the solutions to these problems. To let us successfully capture the body of the knowledge of how to build a good architecture that meets the needs of their users, these solutions and their relationships are codified formally. They form a common pattern language for conveying the structures and mechanisms of our architecture. The primary

reason is not on technology alone but on creating a culture to document and support sound engineering architectures and design as well.

4.1.2 Pattern Languages

The other important concept for design patterns is that of a *pattern language*. In [Smith98] Smith explains that pattern languages are not formal languages, but a collection of interrelated patterns. In other words, a pattern language defines a collection of patterns and the rules to combine them into an architectural style. It describes software frameworks or families of related systems. People cannot only use a pattern individually with a specified solution, but also combine them to realize a whole functionality. Actually, an entire application may be based on a pattern language [Buschmann96], or a combination of several patterns from different areas.

With more experience in using these patterns, the experts will increasingly integrate groups of related patterns to form new pattern languages or expand existing pattern languages. These pattern languages will encompass a family of related patterns that cover particular domains and disciplines ranging from concurrency, distribution, organizational design, software reuse, real-time systems, business and electronic commerce, and human interface design [Smith98].

The pattern language is not just a solution to an individual problem, but to help for integrating patterns in complicated problems in a particular domain. This thesis will discuss common patterns for the e-commerce domain.

4.2 Representing a Pattern

The patterns were conceived as textual based descriptions [Alexa64] in the first moment. Later, graphical elements such as tables, pictures, and diagrams, are increasingly added to the text to represent a solution by the pattern.

Patterns have been rewritten for several times basing on the predefined forms. Up to now there are several pattern forms exist in the literature. For example, the Alexandrian form [Alexa77], the GOF (Gang of Four) form [GoF95], and the Coplien form [Coplien95B]. The pattern-oriented software architecture (POSA) form developed by Buschmann in [Buschmann96] has been employed in software architecture. A new extension of this form, interpreting forces as non-functional requirements and using Non Functional Requirement (NFR) framework to analyze the context and the sequence of a pattern was addressed in [Ong03][Araujo02A].

The following sections will describe the Alexandrian form, as well as the POSA form with the new extension.

4.2.1 The Alexandrian Form

In the book *The Timeless Way of Building* [Alexander79], the architect Christopher Alexander describes the pattern as follows:

Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution.

As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves.

As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant.

Implicitly, Alexand defines a pattern as a construct of three parts. We conclude that Alexandrian form [Alexa77] is composed of three elements:

- Context: A situation that gives rise to a problem.
- Problem: Describes the pulling forces in that context.
- Solution: A proven resolution of the problem.

4.2.2 Structure of Software Design Patterns

Christopher Alexander says, “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice” [Alexa77]. Although Alexander was talking about patterns in buildings and towns, his theory is also used in software design patterns.

Software design patterns attempt to describe successful solutions to common software problems and have been proven useful in helping people reuse in successful practices. In essence, a pattern consists of a statement of a problem with an explanation of the solution in a context. Based on the essential pattern representation, Alexandrian form, a comprehensive pattern representation is presented in [Buschmann96]. It consists of the following elements [Araujo02]:

- Name: A single word or a short phrase to refer to the pattern.
- Also known as: Other well-known names for the pattern
- Example: A real-world example that demonstrates the occurrence of the problem before the pattern is applied.

- Context: The pre-conditions under which the problem and its solution seem to recur, and for which the solution is desirable.
- Problem: A statement of the problem including a discussion of its associated forces.
- Solution: An approach about how to realize the desired outcome based on the pattern.
- Structure: A graphical representation of the classes in the pattern using a notation based on the Object Modeling Technique(OMT) [Rum91].
- Dynamics: Typical scenarios describing the run-time behavior of the pattern.
- Implementation: Guidelines for implementing the pattern.
- Example resolved: Discussion of additional effects after applying the pattern.
- Variants: An additional strategy to make up for the limitation of original pattern in utility.
- Known uses: A well-known example of the use of the pattern.
- Consequences: The outcome the pattern provides.
- See also: Other patterns to solve similar problems.

4.2.3 Rewriting Patterns Using Force Hierarchies

The essential components of each pattern form are: name, problem, context, force, and solution [Coplien95A].

In Ong and Weiss's paper [Ong03], a pattern representation is proposed which explicitly outlines the forces. It suggests an approach for analyzing the text of an existing pattern description, and extracting the force hierarchy for the pattern from it.

Based on the pattern representation in [Ong03], a new pattern form using force hierarchies and explicitly listing the forces "hidden" in a pattern description is explored in this thesis during analyzing a single pattern. In the new pattern representation, we keep the original components and extend some of them, which are: name, problem, context, forces, solution, resulting context, related patterns and example. The forces of patterns are listed by short sentences. A force hierarchy is used to identify the contributions and trade-offs either for a single pattern in "resulting context" part or for a group of related patterns outside of a pattern representation. As the pattern interrelationships are shown in a pattern map in chapter 5, the links in that map are explained in "related patterns" part.

The combination of several force hierarchies of a set of patterns provides a cogent support to obtain the overall contributions and side effects on a system. It allows pattern users to explicitly capture the effect of applying a given set of patterns, and then help them make appropriate choice in consideration of design trade-offs.

As a result, the force hierarchy is also used to compare different patterns through analyzing a set of related patterns under the given context. Where we may apply patterns during design process there are two scopes: application domain and pattern solution. They will be discussed respectively in the next section.

4.3 Application Domain Forces

All patterns exist in a particular context in system design. We find several design contexts, or application domains, such as e-commerce system design, distributed system design, real-time system design, and control system design. Under a specific application domain, the force hierarchy represents the top-level non-functional requirements corresponding to the design context, and the contributions of the associated resources.

In force hierarchy [Araujo02A], the total satisfaction soft-goal is decomposed into one or more forces that describe the design context under analysis. All the resources associated to those forces comprise the next lower level in the hierarchy.

This thesis is discussing patterns in e-commerce domain. Thus, we need to identify the forces that encompass this design context firstly. Table 4.1 shows some forces we have found as of significant impact on e-commerce system design. Then the resources related to these forces are shown in Table 4.2.

Forces and Non Functional Features	Brief Description
Accuracy	Degree to approach precise results.
Attractability	Ability to attract more customers.
Efficiency	Deal with the use of the resources available for the execution of software, and how this impacts response time, throughput and storage consumption.
Maintainability	Ability to provide an adequate framework for problem fixing.
Performance	Extent to which system activities are accomplished promptly
Persistency	Keep states persistent.
Predictability	Ability to indicate what will happen.
Readability	Keep interface neat and readable.
Reliability	Allowable time between failures or the total allowable failure rate.
Reusability	The utility of objects or components across multiple contexts.
Security and Policy	Means protecting information which includes confidentiality unauthorized disclosure and tampering.
Scalability	Ability to provide satisfactory performance as greater demands are placed upon it.
Trust	"Trust is about social relationships, and about building networks that deliver what they promise, be a product, a collaboration, or simply reliable information" [Lynch03]
Usability	Easy and flexible to use.

Table 4.1. Forces and Non-functional features in e-commerce system

Resources	Brief definition
Effort	Human effort required to design and build a system.
Money	Cost to build a system.
Space	Limit on the memory available.
Time	Time taken to respond to an event, time-to-market, medium time between failures.
Computation	Processing that can be done in a unit time (capacity).
State	Information status to capture.
Scaling	Minimization of centralized elements (number of additional users, features, requests, components without sacrificing acceptable performance).

Table 4.2. Resources involved in an e-commerce system design context

4.4 Pattern Solution Force Hierarchy

To construct a pattern language, the first step is to choose the patterns which best describe the overall scope, or application domain of the project [Alexa77]. Then we combine the application domain force hierarchy with the contributions of each pattern to the involved resources to obtain pattern solution force hierarchy.

The total satisfaction soft-goal is on the top of the force hierarchy. The forces corresponding to the design contexts are given at the next level. All the resources associated to those forces encompass the lower level in the hierarchy. This indicates the effects of the application by applying the pattern.

To avoid making the diagrams too busy, only the top-level non-functional requirements and resources actually affected by the pattern are shown in the pattern solution force hierarchy (see Figure 4.1).

Figure 4.1: Force hierarchy for e-commerce system design

4.5 Roles in Pattern Application

In the stage of developing an e-commerce application system, many stakeholders make much contribution to system requirements, domain model analysis and implementation. Considering system analysis and design instead of full development lifecycle, Carlson narrows down those stakeholders to five groups [Carlson01]: consumers (C), business analysts (BA), web application specialists (WAS), system integration specialists (SIS), and content developers (CD). The Figure 4.2 describes the interactions between each group. These five roles can be considered as pattern users. The consequence of each pattern will be in allusion to these five aspects in this thesis.

Figure 4.2: E-commerce stakeholder interactions (adapted from [Carlson01])

4.6 Classification of Patterns

A pattern category is an index of related pattern. A meaningful set of categories for classifying patterns will help pattern users find which pattern they need. Patterns can be classified in a number of ways, such as levels of abstraction, help on similar properties and dependence on a particular application domain. Linda Rising categorizes the currently published patterns in a long list that contains 66 categories in [Rising00A]. These classifications come from the original pattern authors. Nevertheless, pattern categories are not just a list of patterns, there should be a general intent and forces that pattern members collaborate to solve [Alan03]. Different authors gave different categories, there are three well-known categories respectively based on granularity, functionality and structural principles.

4.6.1 Granularity

To develop a software system from basic structure framework to the concrete realization of particular design structure, there are three levels of granularity have been specified:

- **Architectural Frameworks:** Every software architecture is built according to an overall structure principles that are described by architectural frameworks as follows [Coplien95B]:

An architectural framework expresses a fundamental paradigm for structuring software systems. It provides a set of predefined subsystems, as well as rules and guidelines for organizing the relationships between them.

- **Design Patterns:** A software architecture always consists of several smaller architectural units, which are described by design patterns as follows:

A design pattern describes a basic scheme for structuring subsystems and components of a software architecture, as well as the relationships between them. It identifies, names, and abstracts a common design principle by describing its different parts and their collaboration and responsibilities [Gamma93].

A design pattern is less than a complete software architecture or architecture framework, but they are used to structure software systems, so they can be seen as microarchitectures [Gamma93].

- **Idioms:** Idioms concentrate on the detailed realization and implementation of particular design issues.

An idiom describes how to implement particular components (parts) of a pattern, the components' functionality, or their relationships to other components within a given design. They often are specific for a particular programming language.

We can see idiom addresses the language specific pattern strategies. It is possible the same idiom that is useful in one programming language does not make sense in others.

4.6.2 Functionality

This category classifies the patterns for implementing a particular functionality. They are distinguished as follows [Coplien95B]:

- **Creation of objects (Creation):** Patterns may specify how to create particular instances of complex recursive or aggregate object structures.
- **Guiding communication between objects (Communication):** Patterns may describe how to organize communication between a set of collaborating objects that may also be independently developed or remote.

- **Access to objects (Access):** Patterns may describe how to access the services and state of shared or remote objects in a safe way, without violating their encapsulation of state and behavior.
- **Organizing the computation of complex tasks:** Patterns may specify how to distribute responsibilities among cooperating objects in order to solve a more complex function or task.

The patterns in this category are closely related to a particular task or functionality, but it is still hard to restrict the type of each function. Often the pattern to the issue which might access to a object might be in the scope of organizing the computation of complex tasks.

4.6.3 Structural Principles

Depending on certain architecture principles, to realize system functionality, the patterns beyond the previous categories are organized by the principles as follows [Coplien95B]:

- **Abstraction:** Patterns provide an abstract or generalized view of a particular (often complex) entity or task in a software system.
- **Encapsulation:** Patterns encapsulate details of a particular object, component, or service to remove dependencies on it from its clients or to protect these details from access.
- **Separation of concerns:** Patterns factor out specific responsibilities into separate objects or components to solve a particular task or provide a certain service.

Coupling and cohesion: Patterns remove or relax the structural and communicational relationships and dependencies between otherwise strongly coupled objects.

4.6.4 Comparison

The above classification scheme forms a guide that helps designers pick up appropriate patterns for specific applications. However, neither the classification of patterns nor the search for a right pattern is a simple task. The granularity of a pattern is sometimes ambiguous, since a pattern can be used to structure either a complete software system or just a single component or subsystem. Another issue is that a pattern can have more than one structural and functional characteristic. Therefore, the boundary between design patterns and idioms is ambiguous. Each pattern exists in a specific context or an application domain. The patterns addressed in this thesis are about e-commerce domain and the category of them is provided in the next section.

4.6.5 Categories of E-commerce Patterns

The patterns we are mining in this thesis are about e-commerce applications. Based on n-tier application architecture in [Hyatt03], presentation tier, workflow tier, business logic tier, and data tier, we divide patterns into 4 categories: user interface (UI) patterns, navigational (N) patterns, infrastructure (I) patterns, and server-side (S) patterns.

As we know, the boundary between each tier is also ambiguous. Sometimes a server-side pattern deals with the manipulation in front end, for instance the object in session pattern resides in server side, but it holds and keeps the status of information in the client side. In this case, we still treat this pattern as a server-side pattern according to its logic locality in the whole system structure.

Figure 4.3: N-tiered logic model (adopted from [Hyatt03])

The brief description of all patterns presented in the sequence of tiers in this thesis can be found in Table 4.3. The page column indicates which page the current pattern is discussed in the thesis and the type column shows which tier this pattern resides. This follows a common convention used in most pattern books.

Pattern Name	Type	Page	Brief Description
Login [Welie03]	UI	42	Ask user to login by email address and password combination if necessary.
Single Access Point [Yoder97]	UI	44	Set up only one way to log into the system.
Check Point [Yoder97]	UI	46	Provide a mechanism for security check and validation.
Policy Statement [Per99]	UI	48	A policy statement of how the information users provided will be used.
Roles [Yoder97]	UI	49	Define the different permission and access rights to different group of users.
Registration [Welie03]	UI	52	Offer users a registration after first use of the data to avoid reenter the same information.
Offload Work to Client [Weiss03]	UI	54	Offload some functions to client-side scripts and leaves some validation at the server as well.
Premium Content Lock [Welie03]	UI	55	A visual marking used to lock premium content.
Footer Bar [Welie03]	UI	56	The navigation links of whole site and condition of use are shown on footer bar.
Required Field Markers [Per99]	UI	58	Provides a special notation scheme in required fields.
Behavioral Anticipation [Gar97]	UI	60	Indicates the effect and consequence of behavior.
Behavioral Grouping [Gar97]	UI	62	A solution that groups control objects on interface by functionality.
What They See is All They Get [Per99]	UI	63	Puts the most important information always on visible space.
Carrot and a Stick [Per99]	UI	65	A mechanism that withholds sensitive content until users submit more information.
Front Controller [Fowler02]	UI	69	A controller that handles all requests for a web site.

Central Dispatcher [Weiss03]	UI	71	Handle all requests in a single “entry” script that dispatches them to the appropriate handler.
Page Controller [Fowler02]	UI	73	An object that handles a request for a specific page or action on a Web site.
Transform View [Fowler02]	UI	75	A view that processes domain data element by element and transforms it into HTML.
Separate Content From Presentation [Weiss03]	UI	77	Separate presentation from application logic.
Template View [Fowler02]	UI	79	Renders information into HTML by embedding markers in an HTML page.
Language Selector [Welie03]	UI	81	List all available language in its own language at the prominent position.
To the Top [Welie03]	UI	87	Provides a link to the top of the page at position of the list.
Frequently Asked Questions (FAQ) [Welie03]	UI	88	Provide an organized section on frequently asked questions and short answers.
Paging [Welie03]	UI	90	Present the results grouped in pages.
List Builder [Welie03]	UI	91	Present the total list and provide editing functionality on selected items.
Tabbing [Welie03]	UI	93	Divide the data in sections and show one section at a time.
Wizard [Welie03]	UI	95	Step users through the checkout process.
Product Configurator [Welie03]	UI	96	Use a direct and visual version of the configured product for users.
Product Comparison [Welie03]	UI	98	Show a matrix of different products and features.
Plan B [Per99]	UI	100	ALT tag instead of web graphics is used to allow more users to perceive web content.
Easy Undo [Rossi00]	N	104	Returns any specific state of the process instead of backtracking.
Explicit Process [Rossi00]	N	106	Provides a feedback about purchase process through “progress” line or enumerating steps.
Process Feed-Back [Gar97]	N	107	Provides a feedback about operation status.
Opportunistic Linking [Rossi00]	N	112	Provides more related links about new products to capture attention.

Advising [Rossi00]	N	114	Helps customers choose what they want by user profiles, bestseller, products on sale and subsystems. Known as: Product Advisor
News Pattern [Lya98]	N	115	Shows new release information on part of pages to capture the attention of customers. Known as: News Box [Welie03]
Newsletter [Welie03]	N	119	Send users a newsletter about updated content regularly.
Hotlist [Welie03]	N	121	Show a list of most popular items.
Push Communication [Rossi00]	N	123	Provides a push-based communication between user and web store using subscribing, mailing and personalizing site, etc.
Dynamic Configuration Pattern [Lya98]	N	125	Provides a means to perform a selection over a set of options from web.
My Site [Welie03]	N	128	A solution to create user's own web contents. Related Patterns: Customization Window
Customizable window [Welie03]	N	130	Provides a window where selected components can be adapted or clicked away.
Simple Search Interface [Lya99]	N	136	A simple search interface with one field that satisfies a large proportion of users. Known as: Simple Search [Welie03]
Simple Search [Welie03]	N	136	Offer a search functionality consisting of a search label, a keyword and a filter.
Selectable Keywords [Lya99]	N	138	Provides a list of possible keywords according to previous search result.
Structured Answer [Lya99]	N	140	A brief abstract of target page and a taxonomy information space compose of structured answer.
Advanced Search [Welie03]	N	143	Offer a special advanced search function with extended term matching, scoping and output options.
Selectable Search Space [Lya99]	N	145	Provides select category (sub-space) to search.
Search Tips [Welie03]	N	148	Offer help on keywords and on matching options.
Selectable Search Engine [Lya99]	N	150	Add link to an existing search engine.
Search Area [Welie03]	N	153	Combine different kinds of search functionality on a dedicated area.

Catalog Pattern [Fer01]	I	159	A catalog class to be a product collection.
Stock Manager [Fer00b]	I	161	A mechanism that keeps track of stocks and their locations.
Shopping Basket Pattern [Rossi99]	I	162	A mechanism that keeps track of user's selection in navigation by a shop cart. Known as: Shopping Cart [Welie03]
Transition Table [Weiss03]	I	165	A mechanism to define rules for the legal transitions between pages.
Continuation [Weiss03]	I	167	Passes context information for processing a user request by encoding each next action and its parameters within form parameters.
Order Pattern [Fer00]	I	169	A class that requests for a product or service
Shipment Pattern [Fer00]	I	170	Describes the shipment of an ordered product or delivery of a request service.
Order/Shipment Pattern [Fer00]	I	171	Describes the placement and fulfillment of an order.
Shopping Process Pattern [Fer01]	I	173	A process that provides shopping cart, order, checkout and invoice.
E-commerce Site [Welie03]	I	177	Provides a virtual store with basic functions for user to choose and pay for goods online.
Value Object [Fowler02]	I	180	An object whose conceptual identity is based on a combination of values of its properties.
Money [Fowler02]	I	182	Represents a monetary value.
Session [Yoder97]	S	185	A session object holds all variables shared by any objects.
Unique Id [Weiss03]	S	187	Provides a unique session id using cryptographic hash.
Client Session State [Fowler02]	I	189	Stored session state on the client
Server Session State [Fowler02]	S	191	Keeps the session state on a server system in a serialized form.
Server-side Session [Man01]	S	194	Session object at server side holds state information.
Property [Foote98]	S	195	A mechanism to access, alter, add, and remove properties or attributes at runtime.
Database Session State [Fowler02]	S	196	Stores session data as committed data in the database.

Serialized LOB [Fowler02]	S	198	Saves a graph of objects by serializing them into a single large object (LOB) stored in a database field.
Application Server [Man01]	S	202	A program that handles all application operations between users and backend business applications or databases

Table 4.3: The index of patterns for e-commerce application

4.7 Chapter Summary

We started this chapter with a discussion on patterns and pattern languages. This led us to the representation of a pattern from Alexandrian form to a new pattern form using non-functional requirement. As the use of new pattern representation will be applied to the patterns under e-commerce domain in this thesis, application domain forces and pattern solution force hierarchy, as well as a redefinition of the roles involved in pattern application have been individually presented in section 4.3, section 4.4 and section 4.5.

Although there exist various pattern classifications nowadays, only three well-known classifications (granularity, functionality and structural principles) have been addressed, as well as their corresponding liabilities. In consequence, a new classification of patterns of e-commerce domain is proposed based on the n-tier architecture.

5 E-commerce Patterns

This chapter compiles published patterns for e-commerce applications. Most of them are from the papers in PLoP [PLoP] 1997 to 2003, EuroPLoP [EuroPLoP], ChiliPLoP, KoalaPLoP [KoalaPLoP], and Welie's website [Welie03]. Fowler has also documented patterns of enterprise application architecture in [Fowler02]. Since some of them describe the issues of e-commerce fields, they are also gathered in this thesis.

All the patterns are presented individually in this chapter using our new pattern form. The related patterns under the same context are grouped in a table indicating the contributions to the target system. In order to show the interrelationships between the patterns and provide a guideline for pattern selection, a pattern roadmap is provided in the next section.

5.1 E-commerce Pattern Roadmap

In general, patterns are grouped under common aspects and application domain. This leads to pattern catalogs, which are the most common scheme of organizing patterns. Pattern catalogs might be quite helpful as long as the number of patterns is low and their interrelationships and applicability are well defined, but it is still difficult to identify the interrelationships between patterns by using them.

The first attempt to organize and identify pattern interrelationships took place in the conference PLoP'95 [Coplien95B]. Pattern writers wrote the pattern name on the paper, placed these sheets on the floor of the main conference room, and connected each pattern to related ones with string indicating the relationship between them. Thus, they obtained a picture about the pattern and pattern links. This picture is defined as *pattern map* [Buschmann96] or *pattern roadmap*.

Based on the classification for e-commerce patterns in the previous section, we offer a pattern map of e-commerce patterns to present the interrelationship between them as shown in Figure 5.1. There are totally 75 patterns and 106 links between them, including 36 links, which are identified in the original papers. In the current pattern format, we keep the original links and investigate more links between them in accordance with the real web stores and our experience. As a consequence, we may count on this pattern map as a means for supporting pattern selection. The links between each pattern indicate the association type between each other. Standardizing the link types may be a future work.

5.2 UI Patterns

In the UI tier, there are many common issues concerned including:

- How to make an application more interactive and readable
- How to make customers trust your site before shopping
- How to provide a convenient way for shopping online

5.2.1 Security Patterns

For e-commerce applications, privacy and security are the most important issues. However, how to design and implement advanced security techniques is outside the scope of this thesis. This group of patterns we address below will help to resolve common issues on security.

A set of related patterns are grouped in a force table. All the high level soft-goals are displayed in the first row of the table. The pattern name and reference are placed in the first column. The “+” mark indicates positive support, and “--” mark indicates negative support for the corresponding soft-goal. This convention is used for all the following tables.

The patterns shown on the first column in the following table are a series of tasks required when a user accesses an e-commerce website.

The reason why to place “+” or “--” mark in the table is represented in the “resulting context” part of each pattern or in the force hierarchy for a group of related patterns.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistency	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
Login [Welie03]			--	+								+			
Single Access Point [Yoder97]			--	+		--				--		+			
Check Point [Yoder97]			--	--								+			
Policy Statement [Per99]			+								+	+		+	
Roles [Yoder97]			--	+		--	+					+			
Registration [Welie03]			--	+								+			
Offload Work to Client [Weiss03]			+	--		+									
Premium Content Lock [Welie03]					+			+				+			+
Footer Bar [Welie03]			+		+			+	+						+
Required Field Markers [Per99]			+		+			+			+				+
Behavioral Anticipation[Gar97]		+	+		+			+			+				
Behavioral Grouping [Gar97]			--		+			+	+						
What They See is All They Get[Per99]		+			+				+						
Carrot and a Stick [Per99]		+	+												

Table 5.1: Force Table for Security Patterns

Login. The *Login* pattern uses an email address and password combination for users to login to a web application.

Context: Users may frequently access a web application that contains large amounts of data belonging to them.

Problem: How do users identify themselves so that stored data about them can be used in the whole shopping process?

Forces:

- Sensitive information of users, such as birthday, phone number, mailing address and credit card number, should only be accessed by authorized parties or individuals instead of being exposed to the public.
- The system needs to identify users who are using its services.
- Users may be annoyed at having to retype information each time they access system.

Solution: Users are asked to input an email address and password combination when they access the system for the first time. Using the email as a login name is to help track the password by email. To allow users to access free content, the *Login* pattern can be employed only later when it is necessary for them to identify themselves. For example, a user can navigate and pick up any products he wants in the shopping cart until an order is placed. For users who will log in very often, the mechanism of storing status on the client side, for example in “cookies”, is widely used to avoid having to retype username and password.

Resulting Context:

- Extra effort in development may decrease the system efficiency.
- Personal information can be stored in site and be modified by users, so the maintainability is improved.

- Only the users who register in the site can use the resources, which increase the site security.

Related Patterns:

- Users who want to *Login* to the system need to be registered through the *Registration*.
- For security purposes, *Single Access Point* pattern uses *Login* pattern as the sole way to access a system.

Single Access Point. Set up a single way of logging into the system.

Context: Security is a very important issue for a web application. Users may have different rights on different models and subsystems, so they need to be authenticated when they access the system.

Problem: How does the system control the validation of users when there are multiple entrances to an application?

Forces:

- It is easy for users to enter an application through multiple ways.
- An application may be the integration of several subsystems that need to be secured.
- Different login procedures may make duplication in code.
- Multiple entry points to an application may restrict the information provided to the users.

Solution: Set up a single way to access the system, and add a mechanism to decide which sub-applications to launch if necessary. The typical implementation is to create a login screen to gather all information about the user, such as username and password.

Resulting Context:

- Extra effort in development may decrease the system efficiency.
- The single entry may increase network overhead and cause delay so that performance is low.
- Only the authorized users can access the application, which increases the site security.
- It is easy for users to modify their information, thus the maintainability is increased.

Related Patterns:

- *Single Access Point* pattern uses *Login* as a sole “gateway” to access a system.
- To improve the security, it adopts the *Application Server* pattern with *Single Access Point*.
- The requirement to define which process or subsystem is allowed for whom is supported by applying the *Single Access Point* pattern in conjunction with *Roles*.

Check Point. The *Check Point* pattern provides a mechanism for security check and validation.

Context: When users access a web application, they are usually allowed to access only areas in which they are authorized. Thus, the additional security check is necessary to keep unauthorized users from the confidential information. The mistakes sometimes made by users should be handled by the system as well.

Problem: How does an application provide a secure “guard” from break-in attempts by incorporating different organization security policies into the application?

Forces:

- Users authentication and validation become more important.
- The mistakes made by users should be caught or punished.
- Handling lots of occasional errors may reduce the system maintainability because of difficulties in debugging.
- The security check algorithm may be able to be added and modified at any point in the development lifecycle, since the security policy sometimes is not available at the early stage of development.

Solution: An object encapsulating the algorithm for the organization’s security policy is created. It usually keeps track of exceptions and determines which action is to be taken based on the security policy. For example, only three typing mistakes are allowed while entering, otherwise the account will be frozen for a predetermined number of days by security policy. This pattern sometimes is separated into two conceptual parts (authentication and authorization) in the distributed system. However, a further discussion is outside the scope of this thesis.

Resulting Context:

- Complex algorithm needs more development effort and time, and as a result efficiency is down.

- It is hard to modify an object with high complexity, so that the maintainability is weak.
- Only the authorized users can access the application, which increases the site security.

Related Patterns:

- To improve the security, *Check Point* pattern provides a security check algorithm for *Single Access Point* according to a *Policy Statement*.

Policy Statement: The *Policy Statement* pattern provides a statement of how the information users provide will be used.

Context: In web applications, forms are used to gather the personal information of end users. This information, such as mailing addresses, phone numbers and the like, might be considered sensitive by users.

Problem: How do you establish sufficient trust with users so that they are willing to share their personal information?

Forces:

- Users are reluctant to provide their personal information, if they don't know how the information will be used.
- Site owners need the accurate information of end users for business purposes.
- Users might agree to share information under certain principles.

Solution: The *Policy Statement* provides a detailed statement of how the information gathered will be used. The policy statement is always placed in a prominent location on the site. Many policy statements give an option for sharing users' information with other firms. A guideline for evaluating privacy policies was published in [CDT03].

Resulting Context:

- A policy statement makes users trust the use of the resources of site and prohibits non-authorized accesses, consequently the reusability, security and trust are raised.
- Extra evaluation of the policy statement may lead to more design effort.

Related Patterns:

- To improve the security, the *Check Point* pattern provides a security check algorithm according to the *Policy Statement*, implementing the *What They See is All They Get* pattern.

Roles. Defines the different permission and access rights to different groups of users.

Context: Users have different rights to access different areas of the web application. For example, the web master has full rights on all information on the site, while a particular customer only has rights on all products information and his own personal profile.

Problem: How does the application manage user-privilege relationships?

Forces:

- It is hard to customize security for a large number of users.
- The security profile of different groups of users may overlap.
- The frequent modification of a user's security profile is hard to maintain.
- It is difficult to manage the M to N user-role-privilege relationship.

Solution: The *Role* pattern uses role objects to define the permissions and access rights for group of users. The user is assigned a set of privileges when he is logging in. The role is always associated with a job title.

Resulting Context: To improve the security of the system, the following patterns are applied together: *Check Point*, *Policy Statement and Roles*, *Single Access Point*, all using *Login* as a single entry to check the user's authorization for access into the site. From the force hierarchy (Figure 5.2) showing the combination of the patterns, system security and maintainability are improved, but results in lower performance, efficiency, and reliability.

Figure 5.2: Force hierarchy of *Login*, *Single Access Point*, *Check Point*, *Policy Statement* and *Roles*

Related Patterns:

- The requirement to define which process or subsystem is allowed for whom, *Single Access Point* is applied in conjunction with *Roles*, *Login* and *Check Point* patterns.
- *Check Point* uses complex algorithms to check *Role* rights, in accordance with the *Policy Statement*.

Registration. The *Registration* pattern offers users a registration process when users access the system for the first time.

Context: Many e-commerce applications provide personalized services for their customers, such as product recommendations.

Problem: How to prevent users from reentering a large amount of data while using the services?

Forces:

- Site owners need the personal information of their customers in order to provide better service from which site owners may obtain much benefit. For example, the clothes with the favorite style of a user might be accepted over recommendation by system.
- Users would not like to reenter a large amount of data each time they access the site.

Solution: Offers users a way to store their information in the system for later use.

Usually, a registration form is provided for users to fill in. After users complete the registration process, the start page is sent back.

Resulting Context:

- Registration allows users to modify their information so that the maintainability is raised.
- Only the authorized users can access the site, which increases the site security.
- Extra effort in development lowers efficiency.

Related Patterns:

- The users who want to *Login* to the system need to be registered through the *Registration* pattern.

- The requirement for identifying the mandatory fields and premium content which is only provided for registered users, is fulfilled by applying *Required Field Markers* and *Premium Content Lock* pattern.
- The need for supporting client side validation in the registration form is achieved by using *Offload Work to Client*.
- Site owners are interested in seducing users to provide more valuable information during *Registration*, hence the *Carrot and a Stick* pattern is considered.

Offload Work To Client. Offloads some functions to client-side scripts and leaves some validation at the server as well.

Context: Users make interaction with an e-commerce application for shopping. In the standard thin client approach, the user fills in some forms, which are sent to the server for processing, and then the responses are sent back to the user. This may produce performance overhead. Sometimes the user needs immediate feedback for certain actions.

Problem: How does an application provide immediate feedback to users?

Forces:

- Server-side validation is time consuming due to a roundtrip to the server.
- Users need immediate feedback for certain actions.
- Offering all validation script on client side may produce client overhead.

Solution: A strategy is to offload some functions, such as input validation or change propagation, to client-side scripts, and place some validation at the server as well.

Resulting Context:

- Immediate feedback to clients saves response time, so we can say that it is a high performance approach.
- Client-side scripts are hard to be tested, which decreases maintainability.
- The pattern is easy to implemented because of simple structure, thus the efficiency is improved.

Related Pattern:

- The need for supporting client side validation in the *Registration* form and *Login* is achieved by using the *Offload Work to Client* pattern, as well as in *Easy Undo*.

Premium Content Lock. The *Premium Content Lock* pattern uses a visual marking to lock premium content.

Context: Some premium content is only provided for users who have registered in the e-commerce application. Usually non-premium content and premium content are shown together on the page.

Problem: How do users distinguish the free content and premium content?

Forces:

- Premium information is provided only for the registered users.
- Users may click the link of premium content by chance, and a new page may pop up to explain unavailability for the users. This action will consume unnecessary time.

Solution: A visual marking is placed beside the premium content item. In order to make clear which content is locked, the icon of “lock” is always chosen as a marking. When the locked item is clicked by non-registered users, users will be asked to register or login.

The markings are supposed to disappear after the user login.

Resulting Context:

- The visual mark gives users an indication as to which content is available, so it saves navigation time. As a result, performance from customer action and the predictability of site are improved.
- This pattern is easy to use, which improves usability.
- The “Lock” mark, prohibiting non-authorized access, improves security.

Related Pattern:

- To indicate that some contents is available only for users who have already registered through *Registration*, the *Premium Content Lock* pattern is used jointly with the *Footer Bar* pattern.

Footer Bar. The *Footer Bar* pattern provides the conditions for use on a footer bar.

Context: The web site contains much material like copyright or defines special elements of the site.

Problem: How do users know the condition for using a site?

Forces:

- The copyright source of any quoted content needs to be explained to users.
- Some elements have special usage on the particular site.
- An additional page explaining the conditions for usage of the site may waste space.

Solution: Add a footer bar with links to the conditions of use. These links may include copyright statement, disclaimers, privacy statements and other legal matters. A footer bar becomes a standard element of the page template. It is usually shown on each page of whole web site.

Examples: The star in the following website

<http://www.telematicsupdate.com/premium.asp> , is used to explain the usage of the *Premium Content Lock* pattern.

** existing Telematics Directory subscribers still benefit from \$50 discount if they upgrade to Level 3*

home | [contact us](#) | [advertise here](#) | [first conferences](#)
© Copyright 2003 First Conferences Ltd. All Rights Reserved. This document contains original material which is protected by copyright. No unauthorised use of material herein may be made without the prior consent of First Conferences Ltd.

Figure 5.3: Example of website using *Footer Bar* pattern and *Premium Content Lock*

Resulting Context:

- The *Footer Bar* provides a guide for the site, which improves the predictability and readability of site.
- Easy implementation saves development effort so that efficiency is high.
- Simple usage for users makes usability of the site and the performance from customers' action higher.

Related Pattern:

- The conditions for the *Premium Content Lock* pattern is shown on the *Footer Bar*.

Required Field Marker. The *Required Field Marker* pattern provides a special notational scheme in required fields.

Context: In web applications, forms usually are provided to gather user information. These forms contain different fields that require input from end users.

Problem: How does the site owner know that the essential information is captured in a web application? How do end users distinguish between mandatory fields and optional fields?

Forces:

- Users may provide false information or provide as little information as possible.
- Site owners often need precise data to do other investigation and market research.
- Missing some required fields might cause an application error, such as a primary key in a database table.
- The additional “error” page may consume extra time in loading. Therefore, the system performance would be lower.
- Additional symbols may make the screen messy. Changing font and color might be distracting since it is not guaranteed that all users can distinguish color and not all browsers may support a particular font.

Solution: The *Required Field Marker* pattern recommends an approach using a field label. A simple icon such as a bullet, asterisk or checkmark would be a good choice. For forms with few fields, the term “required” can be used as a label. Another option is grouping required fields on one portion of the form. This will make required fields clearly visible.

Resulting Context:

- The approach is easy to develop, which raises the efficiency of system.

- Field labels give users an indication, and save time in filling out forms, hence performance from customer action and predictability are improved.
- Easy usage raises the system's usability and reusability.

Related Patterns:

- The demand for a behavioral indication is supported by adopting the *Required Field Markers* pattern jointly with the *Behavioral Anticipation* pattern and the *Behavioral Grouping* pattern usually in *Registration* process.

Although the *Behavioral Anticipation* and *Behavioral Grouping* patterns were first introduced as hypermedia patterns by Garrido [Gar97], they are widely used in e-commerce applications, so they are considered as e-commerce patterns. These two patterns were discussed under a windows application context in the original paper, while they will be rewritten for an e-commerce domain in this thesis.

Behavioral Anticipation. The *Behavioral Anticipation* pattern provides a mechanism to indicate the effect and consequence of any behavior.

Context: For web applications, data is exchanged between end users and systems by users' behavior at the client side.

Problem: How do you indicate the effect and consequence of the client side behavior?

Forces:

- Many different active objects are provided to users.
- Users may be confused about what to select.
- Graphic icons could not provide clear indication of what will happen and they could distract the users' attention.

Solution: Provide differentiated feedback about the effect of the users' behavior. There are many optional approaches providing feedback: highlighting, color changes, blinking text, and different icons for the cursor. Placing a short text-based explanation in a small area is another way.

Known Uses: A well-known example is that a destination URL is shown on the status bar before clicking a link on a web page.

Resulting Context:

- The pre-indication of action might attract users, therefore the attractability and predictability are increased.
- The easy implementation saves development time so the efficiency is increased.
- Pre-indication avoids unnecessary actions, thus the performance of customer action is improved.
- The independent elements raise the reusability of system.

Related Pattern:

- To support the identification of user interaction, the *Behavioral Anticipation* pattern, a kind of *Explicit Process*, is always used in conjunction with the *Behavioral Grouping* and *Process Feed-back* patterns, with the aid of the *Offload Work to Client* pattern.

Behavioral Grouping. The *Behavioral Grouping* pattern groups control objects on interface according to functionality.

Context: Many different kinds of control objects, such as anchors and buttons, are used on pages in web applications.

Problem: How do you organize different types of control objects so that users may easily understand the meaning of operations?

Forces:

- Different kinds of control objects may provide different functionalities.
- A variety of control objects may make pages messy.
- The variety of functions and diversity of tasks does not allow solutions based on simple conventions such as “a back button is always on the right”.
- Placing control objects at random might interfere with substantive information.

Solution: Group control objects in a specific screen area according to their functionalities, and make each group perceivable in a different place.

Resulting Context:

- Elaborate interface design may need more development and design effort so that it brings negative effect on efficiency.
- Grouping objects in a perceivable area will give users an indication and save navigational time, thus the performance, predictability and readability are raised.

Related Pattern:

- To improve the predictability of pages, the fields with *Required Field Marks* are grouped in a particular place by applying *Behavioral Grouping*.

What They See is All They Get: The *What They See is All They Get* pattern puts the most important information always on visible screen area.

Context: Forms are used to gather user information in web applications. These forms may contain many fields.

Problem: How do developers ensure users are able to see everything on the web form via different browsers?

Forces:

- Users may be reluctant to scroll down long pages.
- Generally, end users have different screen resolution settings according to their preferences. The minimum screen resolution is 640 pixels by 480 pixels, however, the only guaranteed screen space is 535 pixels by 295 pixels since some essential interface elements, such as menu bars and title bars, take up some space [Lynch97].
- Error pages indicating incorrect usages might reduce customer's interest and cause unnecessary delay.

Solution: Display the most critical page elements, such as entry fields and push buttons on the upper left space, since these elements are key points to continue using the application. Keep the most important information within the area in 535 pixels wide and 295 pixels long. Try to keep the form style consistent, for instance, always place submit button in the same place.

Resulting Context:

- Keeping the most critical elements always on a visible area may attract users, and it improves the readability as well.
- The perceivable elements save navigational time, thus performance from customer action is improved.

Related Patterns:

- The *Policy Statement* pattern implements the *What They See is All They Get* pattern.

Carrot and a Stick: The *Carrot and a Stick* pattern provides a mechanism that withholds sensitive content until users submit more information.

Context: Filling in web forms is a very important method to gather the information on end users in web applications. Some forms include users' personal information that they may be unwilling to provide.

Problem: How do you make end users provide information that they are reluctant to share?

Force:

- Users are reluctant to share their personal information, and so navigate to other sites where free content is provided, or subvert required fields with false information [GaTech98].
- Site owners need as much information about their end users as possible to build mailing lists, generate users requirements, or justify the expense of web site maintenance.
- It is difficult to determine the appropriate incentive of an anonymous user.

Solution: In order to “push” users to share their personal information, only a portion of content that users are eager for is offered before requesting personal information. The whole content, called the “stick”, is withheld until the requested information is provided.

Resulting Context:

- Only a portion of content is provided, which raises the site attractability.
- Easy implementation saves the development effort and improves efficiency.

Related Patterns:

- In order to make users provide more valuable information, the *Carrot and a Stick* pattern is used in *Registration*.

This section documents the patterns according to the sequence of tasks when users access an e-commerce application. If it is the first time for the user to enter this system, a *Registration* form is provided to retrieve the user's personal profile. Otherwise, the user can *Login* to obtain the permission to enter the application. In consideration of system security, *Single Access Point* provides the only way to access the system by *Login*. A complicated algorithm of *Check Point* is applied to allocate different resources to the user depending on the *Roles* and the organization's *Policy Statement*. As some content is more important, such as the *Policy Statement* and the *Registration form*, they should be displayed in a prominent place where everyone is able to see which is guaranteed by *What they see is All they Get* no matter what the client's screen resolution is.

During the *Registration* process, some patterns help user fill in the forms with few mistakes. For example, *Required Field Markers* indicate what fields are mandatory or not. The *Offload Work to Client* pattern implements client-side validation as well as server-side validation. The *Premium Content Lock* pattern avoids a casual clicking event on an unavailable link that may pop up a new page. In such a case, the system may ask for *Registration*. Sometimes the user is reluctant to provide personal information, therefore *Carrot and a Stick* withholds sensitive content until he submits more valuable information. After the user finishes filling in the registration form, he would submit it by a button. *Behavioral Anticipation* provides a tool-tip "submit" when the mouse is on the submit button.

There are many different types of UI control objects on web pages, providing different functionalities. *Behavioral Grouping* groups them according to functionality in global, contextual, structural and application objects, and places each group perceivable in different screen areas. The submit button is always placed on the same place in each form. The use of *Premium Content Lock* is always explained on the *Footer Bar*.

The following figure 5.4 is the force hierarchy of patterns used in a registration process. As the contribution links indicate, applying these patterns improves efficiency,

security, usability, time performance, predictability, readability on pages and attraction to customers, but has negative effects on maintainability.

Figure 5.4: Force Hierarchy of Patterns in registration processes

5.2.2 Page Content Patterns

This section will discuss the patterns about the content of web pages.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistency	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
Front Controller [Fowler02]			-	+		-							+		
Central Dispatcher [Weiss03]			-	+		-	+								
Page Controller [Fowler02]			+	--		+			+						
Transform View [Fowler02]			+	+		-					-		+		
Separate Content From Presentation[Weiss03]			+	+		-					-		+		
Template View [Fowler02]			+	--		+							--		
Language Selector [Welie03]			--	+		--			+					+	

Table 5.2: Force Table for Page Content Patterns

Front Controller: A controller that handles all requests for a web application.

Context: In a complex web application, many issues, for instance authentication, internationalization and particular views for certain users, need to be considered when you are handling a request.

Problem: How do developers handle the different requests from front side efficiently for a web application?

Forces:

- If a controller handling front requests is scattered across multiple objects, the implementation will be duplicated.
- It is difficult to change behaviors at runtime.
- Creating controllers for each request might weaken the performance and scalability of a system.

Solution: Consolidate all request handlings by channeling requests through a single handler object, which can be modified at runtime. A front controller usually has two parts: web handler (a handler for request at front end) and a command hierarchy. The web handler is always implemented as a class instead of a server page since it does not produce any response. It just decides which command will be chosen. The commands are also classes rather than server pages although they are often passed the HTTP information.

Known Uses: A well-known example is Struts.

Resulting Context:

- A single handler object with sophisticated logic needs extra effort in development, thus the efficiency is decreased.
- Indirect access to the command may decrease the system performance.
- The structure of separating command from handler makes an application with complex logic easy to modify, but hard to test.

- The independent commands make code scalable.

Related Patterns:

- When designers create a web application, they might be interested in guaranteeing a desirable degree of scalability, *Front Controller*, with the same mechanism as *Central Dispatcher*, is used together with *Transform View*.

Central Dispatcher. Handles all requests in a single “entry” script that dispatches them to the appropriate handler.

Context: After defining the detailed navigation logic in a *Transition Table*, you need to define the user requests for each page, which may require special pre- and post-processing (such as logging, authentication and output transformation). The common approach is to write separate scripts for each type of request, which is dispatched by the web server to the appropriate script according to script names. Although this solution is simple, it may result in duplicated code since the scripts have to handle navigation and presentation logic at the same time.

Problem: How does an application handle user requests efficiently?

Forces:

- Separate scripts for each request may result in duplicate code.
- Encapsulating all scripts into one is hard to test.

Solution: This strategy is to centralize the processing of user requests by handling all requests in a single “entry” script that dispatches them to the appropriate handler. Thus, the codes for pre- and post-processing are both located in the central dispatcher. A clear separation between the code determining the action to handle and the code handling the request should be taken into account when applying this pattern.

Resulting Context:

- This pattern is more complex than individual scripts for each user action, so it may need more development effort and time.
- Centralizing requests makes an application with complex navigation logic easier to maintain since the navigation and presentation are separated.
- Locating the pre- and post-processing in the central dispatcher may ensure the consistency of behaviors from different pages.

- Compared with individual script for each request, the performance is lower because of a longer response time.

Related Pattern:

- When designers create a web application, they might be interested in guaranteeing a desirable degree of scalability, *Central Dispatcher*, with the same mechanism as *Front Controller*, is often used together with *Separate Content From Presentation*.
- The need to keep process consistent is satisfied by exploring *Continuation*.

Page Controller: An object that handles a request for a specific page or action on a web site.

Context: Sometimes complex business logic is implemented in an e-commerce application. When you send a request on a static HTML page, the server will generate another page with the dynamic information.

Problem: How do developers handle a particular action from a page?

Forces:

- The requests from each page need to be handled.
- When controller logic is simple, *Front Controller* may make overhead.
- It is difficult to completely split out HTML from business logic and it will take more design and development effort.

Solution: This strategy uses an object controller to handler requests for each action. The controller may be the page itself in a server page environment, or a separate object that responds to that page. Simply, the controller is tied to each action, such as clicking a link or a button in a web application.

Known Uses: The Page Controller has been implemented using scripts (CGI, Servlet, etc.) or server pages (ASP, PHP, JSP, etc.)

Resulting Context:

- The approach of mixing logic and view together is simple to implement, thus the efficiency is improved.
- The interdependency of view and logic make system reusability and maintainability lower.
- Directing access resource shortens the response time, consequently the performance is raised.

Related Patterns:

- The requirement for efficient development and saving time is supported by applying *Page Controller* combined with *Template View* in sever pages.

Transform View: A view that processes domain data element by element and transforms it into HTML.

Context: Data is transferred between end users and the system in a complex web application. Domain-oriented data is exchanged with a database at the back end. Users view data in HTML format through browsers at the front end.

Problem: How do developers convert domain data to HTML?

Forces:

- Data is stored in a universal format called XML (Extensible Markup Language), which is flexible, allowing the exchange of a wide variety of data on the web.
- Web pages are rendered in HTML format, a non-proprietary format accepted by all browsers.
- Combining business logic and view on the same page makes code hard to maintain.

Solution: Create a program that converts domain-oriented data to HTML. This program, called a transformer, can be written in any language; however the dominant choice is XSLT. XML is passed to an XSLT engine. The logic for the transformation is captured in an XSLT style sheet that is also applied to the transformer. The transformer employs this style sheet with the XML to yield the output HTML that be viewed by HTTP.

Resulting Context:

- The independent data models make code portable.
- Separation of the presentation and data model makes code easy to maintain and helps manage work division.
- Since this strategy depends on a specific scripting language, such as ASP or JSP, to apply a template processor to a data model, it decreases reusability.
- Indirectly manipulating the data model may bring performance overhead.

Related Patterns:

- In creating a web application designers might be interested in guaranteeing a desirable degree of scalability, so *Transform View*, with the same mechanism as *Separate Content From*, is used together with *Template View* and *Front Controller*.
- To support multiple languages, *Template View* is usually used together with *Language Selector*.

Separate Content From Presentation. A strategy to separate presentation from application logic

Context: An e-commerce application should support multiple types of web clients with different presentation requirements, such as PDAs or Internet TV. Thus, during development the designer keeps getting changed to function on different web clients.

Problem: How does an application support multiple types of web clients without modifying output format?

Forces:

- Support for multiple types of web clients is always an afterthought.
- Presentation and application logic are often interwound.
- The unclear division of roles between web designer and application programmer makes the separation between presentation and application logic complicated.

Solution: This strategy defines an intermediate content representation as an interface between presentation and application logic, and then uses CSS, template, or XSLT to convert such content to the target output format. As presentation based on XML is independent of the scripting language, the XML presentation of the content can be mapped to the particular output format required by the different browsers.

Resulting Context:

- Template is faster than XSLT, while XSLT is more efficient in implementation.
- The structure separating content from presentation makes content modification easier.
- Compared with the structure that combines content and presentation, the performance here is lower due to longer response time.
- To apply a template processor to an internal data structure is dependent on a specific scripting language, thus decreasing reusability.

- CSS can separate content and formatting, but can't change the structure of the content. However, template and XSLT can generate an output format for any target, which improves the scalability.

Related Patterns:

- In creating a web application we are interested in guaranteeing a desirable degree of scalability, therefore *Separate Content From Presentation*, with the same mechanism as *Transform View*, is often used together with *Central Dispatcher*.

Template View: Renders information into HTML by embedding markers in an HTML page.

Context: A complex web application contains static information as well as a dynamic data exchange, including login, registration, and transaction.

Problem: How do developers make dynamic HTML pages that display requests from clients?

Forces:

- Static HTML pages cannot satisfy the requirement of an e-commerce application, since users can't place an order over it.
- Combining business logic with HTML pages makes the system unmanageable.

Solution: Embed markers into a static HTML page. All programming logic is placed in a helper object, which simplifies the page. When this kind of page is used to service a request, the markers will communicate with a real program (helper object), and be replaced by the results of some computation such as a database query.

Resulting Context:

- It is easier for most developers to use and to learn, and thus it saves development time.
- It decreases maintainability since embedding markers into HTML pages is hard to test.
- Compared with the structure of splitting logic into multiple views, the scripts shorten response time for users.
- Separating programming logic from template viewers helps distribute work. Non-programmers work on the content of viewer, and programmers work on components with business logic.

Related Patterns:

- The requirements for efficient development and saving time are supported by applying the *Template View* pattern combined with the *Page Controller* and *Transform View* patterns in sever pages.

Language Selector: List all the available languages in their own spelling.

Context: Globalization and localization are the most important features of a successful e-commerce application. It should allow people with different native languages to use the services provided by the application.

Problem: How does an application provide users' preferred languages?

Forces:

- A successful e-commerce application should provide different languages for wide use.
- Language should be localized so that the local user is able to recognize the content.

Solution: Lists all the available languages in their own spelling. The original author recommends approaches such as links to language options or a combo-box as a language selector [Welie03]. Another popular implementation is to automatically show the local language according to the settings of system, so that the users don't need to select a preferred language. The content of pages is always placed in the XML data model, which is transferred by XSLT into HTML.

Resulting Context:

- Web developers who know different languages are required.
- Separation of presentation and data model structures makes the site easy to maintain.
- Indirectly manipulating the data model may cause extra time.
- Providing a user's preferred language may improve the scalability and readability of the site.

Figure 5.5: Force hierarchy for *Language Selector*

Related Patterns:

- To support multiple languages, *Template View* is usually used together with *Language Selector*.

This section has discussed the patterns manipulating the page content. With the same mechanism as *Central Dispatcher*, *Front Control* uses one controller to handle all requests for the whole site. The web handler decides which commands will be chosen. It is flexible to add more commands, but need extra effort for the complicated programming logic. The *Page Controller* pattern only handles the request for a specific page or action using script language. Compared with *Front Control*, the *Page Controller* makes process faster due to its direct access. However, the scalable structure of *Front Control* has closed the performance gap to *Page Controller*. The *Central Dispatcher* pattern decouples the presentation and navigation, which centralized the scripts handling navigation into one script file. The force hierarchy for Page Controller, Front Controller and Central Dispatcher is shown in Figure 5.6.

Figure 5.6: Force hierarchy for *Page Controller*, *Front Controller* and *central Dispatcher*

The *Template View* embeds arbitrary programming logic into pages to render information into static HTML. Such embedded scripts perform poorly as program modules. However, *Transform View* processes domain data by elements and transform them into HTML. The separation of the presentation and data model makes code easy to maintain, and helps the project manager to decouple the roles of application developer and web designer, which needs be traded off on performance. *Separate Content From Presentation* uses the same mechanism as *Transform View*. Their contributions are shown in the Figure 5.7 below.

Figure 5.7: Force hierarchy for *Template View* pattern and the *Transform View* pattern

5.2.3 Page Elements Patterns

This section will address the patterns on page elements.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistency	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
To-the-top Link [Welie03]					+			+							+
Frequently Asked Questions (FAQ) [Welie03]			--	--	+				+						+
Paging [Welie03]					+			+	+						
List Builder[Welie03]			--			+	+		+						
Tabbing [Welie03]			--		+				+						
Wizard [Welie03]			--		+			+	+						
Product Configurator [Welie03]		+	--		+	--									+
Product Comparison[Welie03]		+	--	--	+				+						
Plan B [Per99]			+			+			+					+	+

Table 5.3: Force Table for Page Elements Patterns

To-the-top Link: Provide a link to the top of the page

Context: Sometimes a large amount of information is displayed on one web page causing the page to be very long. Users have to scroll down when the page is longer than one screen.

Problem: How do users go to the top of the page?

Forces:

- Splitting a long page into several pages makes content inconsistent in special situations, such as a FAQ page.
- Users may be annoyed by scrolling up and down on a long page.
- It will take some time to go the top of a very long page using the scroll bar.

Solution: Provide a link directly to the top of a page. This solution helps users go to the top of the page or to the main navigation object in the shortest way. Usually a link with a label such as “to the top” or an up-point icon is placed in position. The link itself points to an anchor placed at the destination you want to reach.

Resulting Context:

- The link pointing to the specific position shortens the time for navigation, thus it improves the performance from customer action.
- A label served as an anchor provides direction, which raises the predictability of a page.
- It is easy to use by just clicking the link, thus the usability is positively supported.

Related Patterns:

- A fast way to go to the specific position or the top of page in *Frequently Asked Questions* is realized by applying *To-the-top Link*.

Frequently Asked Questions (FAQ): Provide an organized section or page on frequently asked questions and short answers.

Context: An e-commerce web site always provides services or products for their users. The users may have many questions about how to use services or purchase a product.

Problem: How does a web site answer the questions that are of interest to users?

Forces:

- It is impossible to answer all questions for each user by mail.
- It will waste space to list all questions on the page, since there might be duplicate ones or individually concerned ones.
- Users may lose interest on the site if they could not get the answers they are concerned about.

Solution: Place frequently asked questions and short answers on a specific section or page. A FAQ is usually put on one page that starts with questions, numbers or categories. The questions are shown by links that link to answers at the bottom. Starting with a list of the questions makes it easier to find a specific question. Sometimes, category is defined by the format “category No.” + “question No.”.

Resulting Context:

- The questions need to be updated frequently in consideration of users’ requests, and thus it is hard to maintain.
- The format “category No.” + “question No.” makes pages more readable and predictable.
- The user may find what is of concern in the list of questions, which may save time on waiting for a reply from an email request.
- Applying this pattern may save time for representatives answering the same question hundreds of times.

- This pattern is easy to implement.

Example:

The Figure 5.8 shows an example of FAQ pattern used in www.futureshop.ca :

Figure 5.8: Example of website using the *FAQ* pattern

Related Patterns:

- A fast way to go back to the list of questions or the top of page in *Frequently Asked Questions* is realized by applying *To-the-top Link*.

Paging: Presents the content grouped in pages.

Context: A large scale of information is presented on web pages. Sometimes this information is grouped together so that users may find the desired one easily.

Problem: How do users go through a large list of items?

Forces:

- Content is separated on different pages when the number of items is too numerous to fit on one page. But it may be annoying for users to seek through many pages using the anchor “next” and “back”.
- To place all items on a very long page is also inconvenient since users navigate only by scrolling up and down.

Solution: Display the information grouped in pages with a fixed number of items. This pattern allows users to jump to any page by the link “previows”, “1,2,3....”, “next”, or by an editable textbox that allows users to input the number of a certain page directly.

Resulting Context:

- It allows users go to any page faster just by selecting the page label, therefore the performance from customers actions is improved.
- The page numbers give users an indication of the position to go, as a result the predictability and readability of pages are raised.

Related Patterns:

- When the list of items is longer than one page in *List Builder*, the *Paging* pattern will be applied with *Tabbing*.
- *Frequently Asked Questions* sometimes uses *Paging* to show more questions in separate pages.

List Builder: Present the total list with editing functionality.

Context: Users select desired products after browsing an on-line store.

Problem: How do users build up and manage the items they select?

Forces:

- Users need to review the items they have selected before placing an order.
- It is necessary to allow users to add, modify, and delete the items.

Solution: Present the total list with editing functionality next to it. If there is no item selected, the status, such as “no item”, should be indicated. There are two common types of implementation: the one is just showing the items with an editing function below each item; another solution is to place an editable textbox and a button next to each item so that users can modify the quantity of each item by changing the number in the textbox. The button is used to delete or show the details of the current item.

Resulting Context:

- Complex logic needs extra design and development effort, so the efficiency is decreased.
- It shortens the total calculation when placing an order, thus the performance is improved.
- All the items are stored in a list, consequently it increases the persistency of system.
- A list shows the all information for the selected items, and it improves the readability of pages.

Example: The Figure 5.9 shows an example of a website applying List Builder. A user can delete the current item with the “delete” button, and modify the quantity of a selection by changing the number in the “Qty” textbox.

Figure 5.9: Example of website using the *List Builder* pattern

Related Patterns:

- When the list of items is longer than one page, the *Paging* pattern should be taken into account.
- The demand for an overview of all selected products and a function for modification is achieved by employing *List Builder* as a “basket” in *Shopping Basket*.

Tabbing: Divide data in sections and show one section at a time.

Context: A large amount of information is shown on web pages. This information usually is grouped in categories. A good website should make it easy for users to find out the information they need.

Problem: How do users pick the section of a site in which they are interested?

Forces:

- Site developers need to present all items on one page so that users are able to obtain a guideline to the whole site at first glance.
- Users want to be able to switch to any other section while navigating.
- Presenting too many sections on one page may make the page complex.

Solution: Divide the data in sections and present one section at a time. This pattern shows all section labels at the top of a page, and lets users switch between sections just by clicking section labels. These sections are typically categorized by labels that are on a discrete scale. When a label is selected, a particular page is shown and the current activated category is highlighted. The categories are usually placed horizontally. Consequently, the number of categories should not be too large. If the number of the categories is greater than 10, an alphabetical index is used.

Resulting Context:

- The complex logic used in categorizing items needs more development and design effort, thus efficiency is reduced.
- Tabs showing the different classifications of items save users' navigational time and provide basic information, accordingly performance from customer action and the predictability of pages are augmented.

Related Patterns:

- When the content of each section is more than one page, *Paging* is used together with *Tabbing* to separate them into multiple pages.

- The demand to provide a navigation widget to indicate steps in a *Wizard* is obtained by applying *Tabbing*.

Wizard: Step users through a checkout process.

Context: There is a serial of tasks to complete when users shop through an e-commerce application. The users may not be familiar with the steps.

Problem: How does a system indicate to users the steps in the checkout process?

Forces:

- In order to achieve a single goal, several subtasks need to be completed by users.
- Users are not familiar with the steps when they access a site at the first time.
- Complex and misleading steps would make users give up in the middle of the process.

Solution: Take users through an entire task at a time. Lead users through each step and indicate which steps left and which have been completed. Users cannot start the next task before completing the current one. Feedback is provided indicating which steps haven't been finished, for example by disabling a navigation widget. The user is able to revise the previous steps by navigating back to a previous task.

Resulting Context:

- This pattern gives users a persistent process.
- The steps in wizard let users know which steps have/haven't been completed, which improves the predictability and readability of the site.
- The elaborate interface design may need more effort in development.

Related Patterns:

- The demand for providing a navigation widget to indicate steps in a *Wizard* is obtained by applying the *Tabbing* and *Paging* pattern.

Product Configurator: Use a direct and visual version of a configured product for users.

Context: A good e-commerce application should help users make decisions as they navigate on a site.

Problem: How does application help users configure the product they may intend to buy?

Forces:

- Different product configurations may attract different users' attention, since some users may be interested in products with preferred configurations, such as color.
- Dynamic configuration of products will personalize a site and attract more eyes to the site.
- Complexity of implementation will extend development cycle.

Solution: This strategy allows users to configure a product using a direct and visual version of a configured product. To configure a product, there may be several steps because of several aspects of the product, such as color, material, writings and the like. Usually a user starts with a base-configured model that can be changed to their preferences.

Resulting Context:

- The functionality that allows users to configure a product may enhance the site's attractability and shorten a user's time in making a decision on production selection. Consequently, performance for customers is raised.
- The complicated UI design and programming logic need more effort and time, which decreases efficiency.
- Loading a list of product need more response time, so the performance is decreased.
- Directly and easily configuring a product using visual tools improves the system usability.

Figure 5.10: Force hierarchy for *Product Configurator* pattern

Related Patterns:

- To show every feature of a product configured, *Product Configurator* is employed conjointly with *Product Comparison*. As the steps in a configuration are independent, *Tabs* can sometimes be applied to set each aspect of a product instead of *Wizard*.
- In order to let users whose client does not support graphs perceive the information, *Plan B* is served to help communication with them.

Product Comparison: Show a matrix of products and features.

Context: When users arrive at a list of products on a page in an e-commerce application, they need to make a decision on similar products with different attributes

Problem: How does an application help users make a decision between similar products?

Forces:

- Similar products are presented on the same section. It is hard for users to perceive the difference in features between similar products by navigation.
- A product with a different configuration may be distinct in price, such as a computer with the same brand but different configurations.
- Only similar products or products with the same features can be compared, for example a book cannot be compared to a car.
- Extra effort is needed to make comparisons on the similar products.

Solution: A matrix of products and features are shown so that users can see the differences in the same feature. If there are many features, they can be grouped so that users can select which ones they want to compare. When there are many products to compare, let users select which ones to compare by marking checkboxes. This should be done in as few pages as possible.

Resulting Context:

- The ability to compare different products might attract customers.
- The matrix which shows the differences in the same features of products avoids writing down the different features to compare, so it save time for customers and makes the site readable for feature differences.
- The complex logic needs more effort in development.
- Frequent changes in products make code hard to maintain and reduce the efficiency of the system.

Figure 5.11: Force hierarchy for *Product Comparison* pattern

Related Patterns:

- To show every feature of a product configured, *Product Comparison*, an instance of *Push Communication pattern* is employed conjointly with *Product Configurator*, which is often implemented after *Search*.

Plan B: A mechanism to communicate the information without graphics.

Context: Graphics, icons and typography are widely used in e-commerce web sites to attract customers.

Problem: How does an application assure that users, whose web client are unable to perceive web page graphics, get information?

Forces:

- The users on low-bandwidth connections may turn off graphic support on their browser.
- Some users may be visually impaired, such as by color blindness.
- Visual appeal may increase the trust from customers.
- Some client facilities such as PDAs are difficult to use to render graphics.

Solution: This strategy creates an alternative by communicating the information without graphics. It can be manipulated using ALT tags to specify a text equivalent in the HTML page. An icon can also be used to communicate the same information instead of font color.

Resulting Context:

- This pattern is easy to employ by adding alternatives such as ALT tag or icons.
- Displaying text or small icons is faster than graphics.
- Replacing graphics for those clients without graphic support may increase the readability and trust for site.

Related Patterns:

- In order to let users whose client does not support graphics perceive the information, *Plan B* is served to help communicate with them while using the *Product Configurator* pattern.

This section discusses patterns that manipulate page elements when customers select products on an e-commerce website.

Product Configurator assists users to choose products with preferred color, model, size, wheels, etc. It uses a direct and visual version to allow users to configure a product model through the Internet. The *Wizard* pattern indicates the different configurable aspects to be used in each step. Sometimes the configurable aspects are independent, so we can use *Tabbing* in such situation instead of *Wizard*.

When users have questions about how to use the site or about the products, the *Frequently Asked Questions (FAQ)* pattern provides a section on frequently asked questions and short answers. The *To-the-Top Link* pattern is always used to jump to the place where the list of questions resides, when the page is longer than one screen. It avoids scrolling up and down.

Tabbing divides the data into sections and shows one section at a time. Normally these sections are described by categories, which are placed horizontally on the top of page. This approach limits the number of categories, so that it is only suitable for items with small classification. When the content in one section is longer than one page, the *Paging* pattern would be used in conjunction with *Tabbing*. *List Builder* provides a mechanism to present the list of items with a summary and edit functionality. When the list of items is longer than one page, the *Paging* pattern will be used together with *List Builder*. The *Paging* pattern actually is a subsidiary pattern of *Tabbing* and *List Builder*.

Graphics are one of the most resource draining elements of a web site, especially for an e-commerce web site. However, not all users can view graphics with their web clients. *Plan B* makes communication possible with clients that are unable to perceive graphic.

The Figure 5.12 shows the force hierarchy of patterns manipulating the page elements. As the contribution links indicate, applying these patterns improves performance, persistency, usability, readability, predictability and trust for site, but trades off on the cost of development and maintainability.

Figure 5.12: Force hierarchy for page element patterns

5.3 Navigational Patterns

Web navigation is the most important manner for users to obtain information through the Web by pushing and pulling based paradigms. There are still a myriad of issues existing in this field.

5.3.1 Process indication Patterns

This section will discuss the patterns that indicate the shopping process.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistency	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
Easy Undo [Rossi00]			--	+	+	--	+	+							
Explicit Process [Rossi00]			--	+	+		+	+							
Process Feed- back [Gar97]			--		+	--		+							

Table 5.4: Force table for Process indication patterns

Easy Undo: Provides a mechanism that returns a specific state of the process instead of requiring the user to backtrack.

Context: The process in an e-commerce application may be complex. Users may make some mistakes, such as wrong spelling, or intend to modify the quantity of a product selected in a previous step.

Problem: How does an application allow users to go back to any step without losing data during the process of shopping?

Forces:

- Some processes in an e-commerce application, such as checking out and registering, may be complex.
- Users need to review and undo a previous operation.
- Some information may be lost by using back buttons.
- Extra effort in developing elaborate interface may be needed.

Solution: Undo facilities are provided to help users jump to any state in a process. These facilities extend the idea of backtracking, which keeps any state in a whole process.

These solutions can be divided into two parts, client side and server side. On the client side, script languages (JavaScript, VBScript) or Java Applets can be used. On the server side, we can choose server pages or server components. The successful solutions are ASP/COM, JSP/EJB and CGI/ISAPI. Other special server solutions are based on the particular server application frameworks, such as Borland Delphi's Midsa/Webbroker, which supports http and other Internet protocols.

Related Patterns:

- It is necessary to allow customers return to any stage of the shopping process, therefore the *Easy Undo* pattern is used jointly with the *Explicit Process* pattern to provide a list of all processes.

- The requirement to keep selected products consistent is realized by applying *Session*.
- There is a need to hold a client's state using *Offload Work to Client*.

Explicit Process: Indicates a purchase process through a “progress” line or by enumerating steps.

Context: There are complex processes in an e-commerce application, for instance the checkout process. Users need to fill in many forms in an application, including shipment information, credit information and personal information. During the process, the user may be confused as to which process has been finished and which has not. When they backtrack to previous pages by clicking a back button at random, the session state may be lost.

Problem: How does an application help users understand a shopping process?

Forces:

- Some processes may be complex in an e-commerce application.
- Users may be disoriented by a process, which should be avoided in an application.
- Avoiding an inconsistent state in each process also needs to be taken into account.

Solution: In this pattern, a perceivable feedback about a process is provided. It is implemented by using a “progress” line or by enumerating the steps that indicate where the user is.

Related Patterns:

- It is necessary to allow customers to return to any stage in the shopping process, thus a list of all processes is provided by using *Easy Undo* jointly with *Explicit Process* specializing in *Process Feed-back*.
- To hold the client’s state, *Session* is applied together with *Explicit Process*, to which *Behavioral Anticipation* is related.

Process Feed-back: Provides feedback about the status of interaction.

Context: When a user purchases something over an e-commerce application, there might be many non-atomic operations, such as getting information from a database that might reside on another machine. If there is no predictable hint about the status, the user may feel that the connection is broken or that he made a mistake. In such a case, the impatient user will leave the site.

Problem: How does an application keep users informed about the status of interaction in an e-commerce website?

Forces:

- There might be complex operations that take noticeable amounts of time to execute in response to some simple interface behaviors.
- User may lose patience when nothing indicates what is happening.

Solution: Provides consistent perceivable feedbacks about the status of an operation, indicating progress in the case of non-atomic operations. In some applications, it may be implemented by changing the cursor's icon or by a progress bar, while in another web application the user may be forwarded to another page that shows the state of the progress.

Related Patterns:

- Customers may want to know what has been done and what will happen in processing requests, so we can choose *Behavioral Anticipation* and *Process Feed-back*.
- *Explicit Process* is a special kind of *Process Feed-back*.

As we know, the checkout and registration processes are the most complex parts of an e-commerce application. This section discussed the patterns used in these processes. Some situations have to be taken into account, such as when customer makes a mistake on spelling while filling in a form, or the information on an ordered product needs to be modified.

Easy Undo, allowing users to go back to any previous steps, is always used in conjunction with *Explicit Process* to keep the customer aware of the current state in the process. Working with *Easy Undo*, the *Explicit Process* pattern minimizes the possibility of reaching an inconsistent state. The common implementation is to use *Session* to hold user state. Once the user makes an unavailable operation in the process, the *Process Feed-back* will indicate the failure by popping up a dialog box or new page. We can say the *Explicit Process* is a special case of *Process Feed-back* using a progress line. The example in Figure 5.13 is a combination of *Explicit Process* and *Easy Undo*. The progress line shows the whole process, in which the highlighted steps are already done and the gray steps are not. The “cart” icon indicates the current step.

Figure5.13: Example of website using the *Explicit Process* pattern and the *Easy Undo* pattern

Elaborate interface design and complex logic need more effort in system development, so the efficiency is lowered when applying these three patterns. Easily going to any step to modify cart information improves maintainability. The progress line in *Explicit Process* gives users a good prediction on the current state. Nevertheless, *Process Feed-back* will produce performance overhead by opening a new message page, and server side script in *Easy Undo* will need more response time. Therefore, the performance of the system is decreased, while the performance for customers is improved

by process indication. The Figure 5.14 shows the force hierarchy of the combination of these three patterns. Since the *Easy Undo* has the same contributions as *Explicit Process* except for negative support for performance, the links between *Easy Undo* to other soft-goal are ignored in order not to make the figure too busy.

Figure 5.14: Force hierarchy for *Easy Undo*, *Explicit Process* and *Process Feed-back*

5.3.2 Communication Patterns

This section will discuss patterns on communication between customers and an e-commerce web site.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistence	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
Opportunistic Linking [Rossi00]		+	--	--	+				+						
Advising [Rossi00] Known as: Product Advisor		+	--	--	+				+						
News Pattern [Lya98] News Box [Welie03]		+	+	--	+										+
Newsletter [Welie03]		+	+	--	+										+
Hotlist [Welie03]		+	+	--											+
Push Communication [Rossi00]			--	--	+		+								
Dynamic Configuration Pattern [Lya98]			--		+		+								+
My Site [Welie03]			--		+								+		+
Customizable window [Welie03]			--		+								+		+

Table 5.5: Force Table for Communication Patterns

Opportunistic Linking: Provides more related links about new products to capture users' attention.

Context: Users are shopping in a virtual store via a web application. When they have purchased what they need, during navigation they might become interested in other products beyond their original plans.

Problem: How does an application attract more eyes to the site, even after a user has already purchased what he wanted?

Forces:

- A site owner wants to keep customers in the store after they have bought products in their original plan.
- Meaningless links make a site interface complex.

Solution: The *Opportunistic Linking* pattern provides the links suggesting other products that related to what the customer has bought. These opportunistic links with strong semantics, are dynamically generated and changed day to day according to users selection.

Resulting Context:

- The related links might attract users to other products in site and help users find these products right away, as a result it increases attractability and performance.
- Complex data models are hard to maintain, and make development hard, thus it decreases efficiency.
- Too many links may increase UI overhead so that it may lower the readability of a site.

Examples: The Figure 5.15 is an example of website www.amazon.com , which uses the *Opportunistic Linking* pattern. When a user picks up the book “Design Patterns”, the

links to books that other customers have bought are displayed below. Other books related to “Design Patterns” are listed. It gives more choice to the user.

Figure 5.15: Example of website using *Opportunistic Linking* pattern

Related Patterns:

- *Opportunistic Linking* is a kind of *Advising* pattern.

Advising: A mechanism to help users choose what they want by user profiles, bestsellers, products on sale and subsystems.

Context: Through an e-commerce application, users look for the products they would like to buy. They have to find what they want from thousands of products by product indexes or search engines that might not offer a complete solution.

Problem: How does an application help users find the product they intend to buy in a virtual store?

Forces:

- Site owners want to induce more users to consume on site because of commercial benefits.
- Some mechanisms such as search engines and indexes are useful but incomplete.
- An application needs to take into account what users need.

Solution: This strategy introduces a specific functionality to help users to choose products according to users' profiles (such as buy histories and preferences), bestselling products, or products on sale. It may be complex to track a user's buying history, since it changes day by day. The implementation of this strategy should follow the global navigational structure.

Resulting Context:

- Specific functionalities are applied to attract users to other products within the site.
- A complex algorithm that allows the application to track a user's buying history may cause extra effort in design and development; hence it lowers efficiency.
- Frequently changing a user's buying history and links may make the system hard to maintain.

Related Patterns:

- *Opportunistic Linking* is a kind of *Advising* that is implemented by the *News* pattern to attract more customers.

News pattern: Shows new release information on part of the pages to capture users' attention.

Also Known: *News Box* [Welie03].

Context: Unlike conventional applications, the navigable space of an e-commerce application is not fixed for the reason that new information is added and existing content is modified frequently. Most websites are tree-structured, so that the larger the information space tends to be, the harder it is for users to navigate.

Problem: How does an application inform the users of new information available or of information updated?

Forces:

- It is impossible to display all new products on the first page.
- Site owners would like to provide new products to customers.
- Additional space for news would destroy the original structure wherein information is organized in items with sub-items.

Solution: A visual announcement of new releases and recent changes is provided.

Usually a specific space with a headline is devoted to the new additional items by linking to corresponding pages. These links should be as short as possible, and be displayed in a noticeable place. Another approach is to add visual clues to the pages in which the content has been added or updated.

Resulting Context:

- Users can get new information by explicit news items instead of navigating through whole site, so that it saves unnecessary navigation. In another word, it improves the performance for customers.
- A specific space holding new updates and changes makes site development easier.
- An organized structure makes a site easy to manage and to be used.
- Frequent changes on products make a site hard to maintain.

- One additional thing needs to be taken into account. For example, when a user accesses a new release page of the Canon PowerShot A80 digital camera through this object, he knows this model is a new one. However, when the user enters this page using navigation through the categories (digital camera – canon-- model PowerShot A80), he might have no idea whether this model is a new one. Therefore, an indication that the product is a new release should be included in each page of new products.

Figure 5.16: Force hierarchy for *News* pattern

Examples: The Figure 5.17 shows example using the News pattern in www.amazon.com, on the right corner, the “What’s New” item lists all new releases. Each product item comes with a short description.

Figure 5.17: Example of website using the News pattern www.amazon.com

Related Patterns:

- *Hotlist and Newsletter* are special cases of the *News* pattern implementing *Advising*.
- *Push Communication* improves the attractability of *News*.
- Sometimes customers may be interested in the items in news section. In this case, *Dynamic Configuration* provides a convenient way to add the current item into a shopping cart.

Newsletter: Sends users a newsletter about updated content regularly.

Context: Although most users do shopping by navigating online through an e-commerce website, not all users have time to surf on the Internet everyday.

Problem: How does an application let users know about the updates and changes on items that they might be interested in?

Forces:

- The content of the site is changed day to day.
- Users need to know the updated content in the situation that they have not much time for surfing online.
- Uninteresting subscriptions might annoy users.

Solution: Send users a regular newsletter about the information they are interested in. When a user registers on the site, a choice on subscription is offered. The newsletter always has the same basic design style as the original site so that it's easy for users to recognize it. In addition to the main content, the newsletter includes the name of company/site, a disclaimer, a date and an issue number. The user may have lost interest in a previous product line, hence it is necessary to allow users to unsubscribe through the newsletter.

Resulting Context:

- Users may get the latest information by newsletters, and it will attract users to shop online.
- Sending a newsletter might be easy to implement, but it is hard to maintain due to frequent changes in the site.
- Users can reach specific pages by links provided in newsletter, and this saves navigation time. Accordingly, performance from customer action is increased.

Related Patterns:

- Usually the *Newsletter* facility is provided during *Registration* by email address, which is a special case of the *News pattern* and is a kind of *Push Communication*.

Hotlist: Shows a list of the most popular items.

Context: Users always select what they might be intent on buying from a large amount of items that provided by e-commerce application. Users might not be interested in all items, but they need to find them one by one before making a decision.

Problem: How does an application help users to find what they are interested in?

Forces:

- An application provides as many products as possible to attract different kinds of users.
- It is hard for users to pick up one item from hundreds of items.
- Users may be not interested in all items.
- Too many items in a list may make the screen appear complex.

Solution: A host list of the most popular items, such as bestsellers, is introduced. The list should not be too large if it is displayed on the first page, otherwise the specific page can be used to show the bestsellers in each category. The most popular one is placed at the top of the list, and is associated with a link to the detailed description of the clicked item. A striking label for the list is used to attract the attention of users.

Resulting Context:

- The facility of receiving the most popular information by *Hotlist* increases the attractability of site.
- Easy implementation by adding links on the site raises efficiency.
- It is hard to maintain because of the frequent changes of information.

Examples: In the Figure 5.18, the “Top Sellers” section at the left corner is an implementation of the *Hotlist* pattern.

Figure5.18: Example of website using the *Hotlist* pattern in www.amazon.com

Related Patterns:

- The *Hotlist* pattern is a special case of the *News pattern*.

Push Communication: Provides a push-based communication between users and the web store using subscription, email and personalization site.

Context: To find a desired product, users have to use the so-called pull-based facilities, for example browsing an index of products, inputting the keyword in a search box, or checking the news list that is provided in most web applications. These facilities require users to enter the store (launch the site pages).

Problem: How does an application help users find what they need even if they don't enter the store?

Forces:

- It may waste a lot of time to develop only pull-based facilities for browsing on the site.
- Finding exactly what they want from thousands of items is very hard sometimes.
- Existing facilities are not complete for helping users.

Solution: A strategy combining pull-based facilities with push-based communication is recommended. This solution can be implemented using subscription, mailing list and personalization.

Resulting Context:

- It is hard to maintain since this pattern relies on a subscription/update mechanism that may change frequently.
- By not having to routinely browse the site, users save time from the regular mailing that gives new information on the site.

Related Patterns:

- To attract more customers, the *My Site* pattern using the *Customizable Window* pattern as a model is chosen together with the *Push Communication* pattern, including the *Newsletter* pattern and the *News pattern*.

- *Product Comparison* is actually an instance of *Push Communication* that has the same mechanism as *Observer*.

Dynamic Configuration Pattern: Provides a means to perform a selection over a set of options from a web.

Context: E-commerce applications should make shopping over the Internet as real as shopping in a physical store. In a web application without the *Dynamic Configuration* pattern, users have to pick up one item by navigation, and put it into a list in one session. In order to buy another product, they need to navigate and select again, and repeat this step to buy more. It is easy to see that this interaction model is too tedious.

Problem: How does an application help users to perform a selection and keep track of products while traversing?

Forces:

- E-commerce applications need to make shopping online as real as shopping in a physical store.
- An application needs to take into account users with low bandwidth and unstable Internet connections.
- Picking up one product per session will take a long time through navigation.

Solution: A mechanism that adds the current displayed product to a list (shopping cart/basket) is recommended. Usually, a button is placed besides the product description. It allows users to shop while “walking” online. Furthermore, validation on the selected items should be performed when the item is put into the shopping cart.

Resulting Context:

- Selecting while traversing reduces a lot of navigation time, and consequently it improves the performance for customers.
- Keeping the state of selections persistent in a shopping cart raises the system’s reliability and persistency.
- It is easy to implement by adding a button/link besides the products listings.

- Additional items on the page and the algorithm tracking item state need more development and design endeavors.

Figure 5.19: Force hierarchy for *Dynamic Configuration* pattern

Examples: In Figure 5.20, the “Add to Cart” button is used to add an item into a shopping cart, which is an implementation of the *Dynamic Configuration* Pattern. The validation is checked before selecting an item.

Figure 5.20: Example of website using the *Dynamic Configuration* Pattern

Related Patterns:

- Sometimes customers may be interested in the items in the *News* section. In this case, *Dynamic Configuration* provides a convenient way to add the current item into a *Shopping Basket* by displaying an “add” button beside the new releases. This mechanism is also available in *My Site*.

My site: The *My site pattern* provides a solution to create a user's own web content.

Context: Many e-commerce applications sell products or services to their customers. The customer might only be concerned about some items or several products instead of all items; for instance a user is usually concerned about the stock that he bought or intends to buy.

Problem: How does the application provide the content of interest to a particular user?

Forces:

- Users may focus on only that part of information that they are interested in.
- In order to attract more users to the site, applications need to provide the information that might be of interest to users.

Solution: A strategy is provided that allows users to create a part of the site controlled by themselves. Users can define the layout and control all elements on their own part of the site by selecting 'modules'. Therefore, only the personalized content is shown for users when they enter the site.

Resulting Context:

- The complex logic in providing this dynamic configuration of the site makes site efficiency lessened.
- Personalized content save navigational time for users, thus the performance from customer interaction is raised.
- The independent modules improve site scalability.
- It is easy for users to build their own site by adding dynamic models, so that as a result the usability of site is increased.

Examples: The Figure 5.21 is the example of *My site*. All the elements are set up by users.

Figure 5.21: Example of website using the *My Site* pattern

Related Patterns:

- The support for multiple languages is achieved by applying *Language Selector* conjunctly with *Transform View*; consequently it improves the trust in site.
- To personalize the site, *Customization Window* works as a model in *My site*.
- The mechanism of adding products to shopping cart supported by *Dynamic Configuration Pattern* is also available in *My site*.
- The requirement for personalizing news and informing users of changes on the site is served by applying *Push Communication*.

Customizable window: Provides a window where selected components can be adapted or clicked away.

Context: Users may only pay attention to the content that they are interested in when they are surfing on a website. It would be attractive to users to provide them particular items.

Problem: How does an application help users to build their personalized area?

Forces:

- Exploring the whole site to find a particular item ends up a waste of time.
- It is ideal to only provide the information that users would like to receive.

Solution: An area like a window with selected components is provided, so that users can build their personalized window by choosing and removing these components. The buttons such as “minimize”, “edit”, “close” and “customize” can be used while setting up the site. In order to keep a window consistent, the settings are stored for each user.

Resulting Context:

(same as the *My Site* pattern)

Related Patterns:

- *My site* uses *Customizable window* as a model to build a personalized site.

Web site owners consider ways to keep customers online longer and improve customers' shopping desire. To capture customer's attention, the *Opportunistic Linking* pattern provides links to products related to ones that the customer already bought. The intent of this pattern is to recommend more products in a comfortable way. Actually, it implicitly improves sales; nevertheless, too many links may cause omission and complex data models. The *Opportunistic linking* pattern is quite similar to *Advising* in pushing customer to buy more products. However, the intent is different. As we discussed above, *Opportunistic Linking* is to keep the customer inside the "store" by giving more choices, while *Advising* is to help them choose what they want. Since *Advising* requires the system to remember each user's buying history, it may need a more complex algorithm.

Compared with a non-news pattern website, the *News Pattern* makes a structure-less (see Figure 5.22) and star-shaped (see Figure 5.23) navigational structure of website more organized, and it shortcuts to information probably underlying a tree-structure site. *Hotlist* and *Newsletter* are special cases of the *News Pattern*, while the *Hotlist Pattern* always shows the most popular items, such as the top 10 bestsellers.

Figure 5.22 Structure-less structure

Figure 5.23 Star-shaped structure

When users register in site, there should always be a newsletter option offered. *Newsletter* regularly sends users a message about the updated content on the website since not all users access the website every day. This mechanism is the so-called "pull" model. The *News* pattern and the *Advising* pattern both belong to the "pull" based strategy. They all require users to "enter" the site, but any new changes may not be of interest to the users. The disadvantage of these patterns is that users may waste a lot of

time navigating the website to find what they really want. *Push Communication* brings up a mechanism combining pull-based methods with push-based communication using subscription, mailing list and personalized site. The *Dynamic Configuration Pattern* provides a solution allowing users to pick up products and add a currently displayed product into a shopping list (cart) by clicking buttons during navigation. It is always used jointly with *Shopping Basket*.

Users surfing over the Internet sometimes want only specific information, such as stock quotes. The *My Site* pattern provides an approach for customers to build their own part of the site to reflect their own interest. It is implemented by picking up a “module” when the customer first logs in. Each module is a *Customization Window*. This mechanism generates more opportunities for site owners to obtain detailed information about their customers, such as their favorite products and habits. *Customization Window* is a special implementation of *Push Communication*. From this valuable information, a site owner may know how to allocate products or probably provide those products on sale for enticing more users to it.

The figure 5.24 shows the force hierarchy of patterns in communication between web sites and users, as discussed in this section. As the contribution links show, employing these patterns improves site attraction, time performance for customer interaction, status persistency, scalability and usability. Nevertheless it has negative effects on efficiency, maintainability and readability because of extra development effort, frequent changes in information and multifarious links on pages.

Figure 5.24: Force hierarchy for patterns in communication

5.3.3 Search Patterns

A successful e-commerce website should be good to help people find what they are looking for. People always get information through navigation on the website. Nevertheless, the bigger the information space becomes, the less desired information users receive. Accordingly, search facilities are provided to return more precise search results. When a user starts searching, there is an information space that is related to his need. A mechanism to measure the performance of search using two ratios: *recall* ratio and *precision* ratio has been addressed in [Lya99]. *Precision* is the ratio of the total number of relevant document (or pages) received, to the total number of documents retrieved, while *recall* measures the ratio of the number of relevant documents retrieved, to the total number of relevant documents. Therefore, the good search facility should be the combination of high precision and high recall. This section will document some search patterns based on this mechanism.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistency	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
Simple Search Interface (Simple Search)[Lya99]	--		--		+	--			--						+
Selectable Keywords [Lya99]	--		--		--	+			+						
Structured Answer [Lya99]	--		--		+	--			+						
Advanced Search [Welie03]	+		--		--	+			--						--
Selectable Search Space [Lya99]	+		--	--	--	+			--						--
Search Tips [Welie03]			+		+			+	+						+
Selectable Search Engine [Lya99]	--		+	--	+	--			--		+				
Search Area [Welie03]								+	+		+				+

Table 5.6: Force Table for Search Patterns

Simple Search Interface: A simple search interface with only one field.

Also Known: *Simple Search* [Welie03]

Context: Although effective search facilities could help users to find accurate search results, users need to perform some interaction such as selecting further keywords.

Problem: How does a search mechanism satisfy users' needs without further interaction?

Forces:

- A simple search may satisfy the needs of users sometimes.
- A simple search may bring out a large amount of options available for users.
- Advanced search facilities are time-consuming since they might require a complicated algorithm.
- An elaborate interface is needed to employ an advanced search.

Solution: A simple, straightforward search interface only including one field is provided. It just implements the simplest search with one step to satisfy a user's needs.

Resulting Context:

- Simple search may retrieve a large amount of options that decrease the accuracy.
- Extra element such as links for advanced search may result in more complex interface that decreases the efficiency.
- Less navigation for simple queries increases the performance for customers.
- Simple queries with a simple phrase or sentence may cause sophisticated computing at back end and need extra response time, and thus it lowers the performance for system.
- Only one field for search may not clear to all users, and thus it will decrease the system readability.
- The simple interface makes implementation easy so that it increases usability.

Figure 5.25: Force hierarchy for *Simple Search Interface* pattern

Related Patterns:

- The need for more accurate search is carried out by exploiting *Advanced Search* displayed beside *Simple Search Interface* to give an additional option for users.
- *The Selectable Search Space* and *Simple Search* with other search facilities compose the *Advanced Search*, which are displayed in *Search Area*.

Selectable Keywords: Provides a list of possible keywords according to the search result.

Context: When users perform a search through a search engine, the most common action is to enter a relevant keyword about the subject of the search. Due to the large search space, there might be many possible items matching a given word. In this case, further keywords might be required.

Problem: How does an application help users to retrieve desired information after the first search?

Forces:

- Users always search over large search spaces.
- Irrelevant keywords may bring undesirable items, hence the application can define relevant keywords for users to improve performance.
- Users need to know which keywords are relevant to the subject of search.

Solution: Based on the search already performed, a list of keywords is provided.

Actually, the keywords can be any combination of the suggested ones. It is implemented by using checkboxes. Users are able to select keywords from a list, since it is easy to make typing mistakes.

Resulting Context:

- A list of keywords may not be accurate for every user.
- Extra development effort in defining keywords may decrease the efficiency and performance of the system.
- Users need to spend extra time in picking up keywords from options, and this step can be implemented only after the first search, so that it decreases customer performance.
- The meaningful keywords improve readability for users.

Figure 5.26: Force hierarchy for the *Selectable Keywords* pattern

Related Patterns:

- To improve the readability and neatness of pages, *Behavioral grouping* is used to arrange search/keyword and the list of selectable topics in a logical unit.
- *Selectable Keywords* specializes in *Structured Answer*.

Structured Answer: Provides a brief abstract of target page and classifies search results.

Context: There might be thousands of relevant links pointing to different pages retrieved in the search result. It is hard for users to pick up a right link from the large amount of ones retrieved.

Problem: How does an application help users to decide on the right link in the search result?

Forces:

- The large amount of search results may include irrelevant items.
- It should be possible to access other items while examining one item.
- The summary of the linking page should be readable so that it is able to help users distinguish the right one.

Solution: A mechanism is provided to help users pick up the most suitable items. The query results include a brief abstract of the target page. The information about links to other possible related ones and sub-spaces are grouped in different categories.

Resulting Context:

- It is not easy to find a desire one from a large amount of links. Accordingly, the accuracy is still low.
- The summary of links increases the readability of search results.
- The algorithm to narrow the results into categories needs more development effort and computing time. Therefore, it may decrease the efficiency and performance of the system.
- Users may easily find a suitable link by summary so that it increases the performance for customers.

Figure 5.27: Force hierarchy for the *Structured Answer* pattern

Examples: The Figure 5.28 below is an example of *Structured Answer* in www.yahoo.com. When entering keyword “LCD”, the list of relevant links with category subspace is shown.

Figure 5.28: Example of website using the *Structured Answer* pattern

Related Patterns:

- The demand for retrieving more accurate search results is aided by exploiting *Structured Answer*, specialized by *Selectable Keywords*, in conjunction with *Selectable Search Space*, as well as *Advanced Search*

Advanced Search: Offers a special advanced search function with extended term matching, scoping and output options.

Context: E-commerce site is an information rich site that contains a large collection of items. Users may not obtain the exact item via a simple search. For users who have deep knowledge about what they are searching for, it is necessary to provide an advanced search mechanism.

Problem: How does an application help users with much knowledge about items to obtain precise results from a search facility?

Forces:

- Simple search ends up with too many options so that users may have no idea as to which is the exact one.
- Advanced search requires users to have additional knowledge of the items that they are looking for.
- Advanced functions offer users control of the search.

Solution: A special advanced search function with extended term matching, scoping and output option is provided. User have more control over the combination of search terms by selecting the keywords “and/or”. It narrows the possible results by defining the conditions for example item types (article, video, book, etc) or item properties (title, date, ISBN, etc). This functionality also allows users to control the output from *Search Results*, such as sorting or *Paging* size.

Resulting Context:

- Extended functionality makes results more accurate, but needs more development effort.
- Complex client interactions make performance for the customer lower.
- Specific search settings save computing time, and consequently it increases the performance of the system.

- Complicated interface controls decrease readability of pages and system usability.

Figure 5.29: Force hierarchy for the *Advanced Search* pattern

Related Patterns:

- The need for more accurate search is carried out by exploiting *Advanced Search* displayed beside *Simple Search Interface* to give an additional option for users.
- To improve the page predictability, *Advanced Search* composed of *Simple Search Interface*, *Selectable Search Engine*, *Search Tips*, *Selectable Search Space* and *Structured Answer*, is displayed in *Search Area*.
- The requirement of raising the search performance is achieved by using the *Catalog* pattern.
- When the search results are more than one screen, *Paging* helps to separate the results into multiple pages.

Selectable Search Space: Specify a category (sub-space) for search.

Context: In order to find the desired information, users perform a search on a web application. It is worth noting that most search engines trade off precision against recall, because it is possible to retrieve more documents by increasing the number of the relevant documents. However, the search precision is still low.

Problem: How does an application perform a search facility with the combination of high precision and high recall?

Forces:

- Users usually search over large search spaces that are mutable.
- An effective search should have high precision and high recall.
- Restricting the search space to known relevant subspaces may improve the search effectiveness.

Solution: A mechanism to restrict the search space into a relevant category is provided.

Variants:

- Users are allowed to pick up one search category at a time. This method requires that any two information spaces not overlap.
- Allow users to combine more than two search spaces together. This strategy is more flexible, but extra navigation is needed to select the desired categories. Thus, this approach should be used only when the previous one is impracticable.

Resulting Context:

- Higher recall and precision rates improve the system accuracy.
- It is hard to define the relevant categories and subspaces, so that extra design and development effort are needed. Consequently, the efficiency is lowered.
- Mutable search spaces make the application hard to maintain.
- More interaction performed by users decreases the performance for users.

- Fewer searches consume less response time and computing resources, therefore it improves the performance of the system.
- Complex UI design and interface controls decrease the readability of pages and usability for users.

Figure 5.30: Force hierarchy for the *Selectable Search Space* pattern

Examples: The following example is implementation of *Selectable Search Space*. The radio buttons and combo-box are used to specify a search space in www.careerclick.com.

Figure 5.31: Example of website using the *Selectable Search Space* pattern

Related Patterns:

- To obtain more accurate search results, *Advanced Search* includes *Selectable Search Space* which is always exploited with *Selectable Search Space* and *Structure Answer*.

Search Tips: Provide help on keywords and on matching options.

Context: Users might enter the keyword/phrase in a form by search unacceptable to the search engine. If this takes place very often, users may lose patience and leave the site.

Problem: How does an application help users input successful keywords/phrases while using a search engine?

Forces:

- A long description of each search may take more UI space.
- Users may make mistakes without any instruction.

Solution: This strategy offers help on keywords and matching options by a short description within one line. Sometimes there is an example beside the search field. For instance, “User quotes to search for an exact phrase”.

Resulting Context:

- The short description is easy to perform and increases efficiency and usability.
- Users easily control the search functionality with help, increasing the performance for their action.
- The description about how to perform searches increases the readability and the predictability of pages.

Figure 5.32: Force hierarchy for the *Search Tips* pattern

Related Patterns:

- In processing complex searches, users may need to know the criteria of manipulating the search facilities; hence *Search Tips* is served with *Advanced Search*.

Selectable Search Engine: Links are provided to other existing search engines.

Context: Through web application, users search on a huge space of information that is far from being categorized and organized. Since this information space is growing at extremely high rates, where the information is added, removed and classified second by second, it is impossible to claim complete information by a single search engine nowadays.

Problem: How does an application help user to retrieve more accurate search results through search facilities?

Forces:

- Powerful search engines consume large amount of money and resource space.
- Search technologies grow at extremely high rates, so the application may need more time to keep track of them.

Solution: This solution provides links to existing search engines while building a web application, so that users are able to select search engines by using a dropdown list displaying the names of all possible search engines instead of accessing the respective websites. In this case, there is one requirement that must be taken into account whether the search services are allowed to be used remotely.

Resulting Context:

- More relevant links will be provided by several search engines. That decreases the accuracy and readability of results.
- Adopting the existing engines reduce the development time and costs, therefore it increases the efficiency of the system.
- Existing engines may be not completely suitable for the task in hand and may not be able to be modified; consequently it decreases the maintainability of the system.

- Users easily access remote search engines by selecting from a dropdown box instead of going to the respective websites; hence it increases performance for users.
- Remote search may consume more response time so that it lowers the performance of the system.
- A search engine is an independent model so that it increases reusability.

Figure 5.33: Force hierarchy for the *Selectable Search Engine* pattern

Related Patterns:

- Customers may require easily captured search facilities, so that they are grouped in a prominent area applied by the *Search Area* pattern. The dropdown list of search engines is usually placed beside the keyword search field.

Search Area: Combine different kinds of search functionality on a dedicated area.

Context: There may be several kinds of search-related facilities in medium to large websites. Although the search functionality is not the main method to access information, it is still an important means to find out desired information.

Problem: How does an application let users perform search facilities easily?

Forces:

- There is more than one search facility provided especially in medium to large sites.
- Putting several search facilities in different areas may make page interface complex and hard to use.

Solution: Group the different kinds of search functionalities in a dedicated rectangular block so that users may reach them directly. Alternative items and helper items are provided beside such an area.

Resulting Context:

- The explicit rectangular block makes search facilities easy to be found. It increases consequently the predictability and readability of page.
- The independent UI element and placement increase reusability.
- Easy implementation increases the usability of system.

Figure 5.34: Force hierarchy for the *Search Area* pattern

Related Patterns:

- The need for easily capturing search facilities is supported by grouping *Simple Search*, *Advanced Search* and *Search Tips* in *Search Area*, which is a special case of *Behavioral Grouping*.

According to the analysis of respective search patterns above, we combine force hierarchies as a whole, excluding the negative support contributions. It is distinctive that the system soft-goals (accuracy, efficiency, performance, readability, predictability and usability) got a significant positive support by combining these patterns.

The *Simple Search Interface* pattern provides a quick and simple solution for searching over Internet. Although it is easy to employ, it may retrieve imprecise query results. Hence, the patterns dealing with these query results, such as *Selectable Keywords* and *Structured Answer*, can be taken into account. The *Selectable Keywords* pattern provides a list of possible relevant keywords so that the search engine may focus on one particular database instead of hundreds of related ones. However, this pattern must be performed after the first search.

Structured Answer adds a brief abstract of target pages along with the links and adopts a taxonomy to organize information space into a meaningful subspace to help users find the most suitable link. *Selectable Search Space* narrows the huge search space to a subspace that saves a lot of query time. As a matter of fact, *Advanced Search* is the combination of the *Simple Search Interface* pattern and other patterns mentioned before. *Advanced Search* needs more client interactions, but it reduces the response time and results in more accurate items. The *Search Tips* pattern provides help on how to use the controls supplied by the *Advanced Search* pattern. As one website is not enough to claim complete knowledge, *Selectable Search Engine* allows search in remote website. All these search patterns can be grouped in a perceivable area as the *Search Area* pattern says, but the restriction of usage should be considered. Therefore, the combination of these search patterns is a proposed solution for building an e-commerce website with powerful search capability.

Figure 5.35: Force hierarchy for several search patterns

5.4 Infrastructure Patterns

5.4.1 Process Patterns

How does a user place an order after searching and making a decision? This section will document the patterns during the process of “buy”.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistence	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
Catalog Pattern [Fer01]		+	--	:	+	--					+				
Stock Manager [Fer00]			--	:	+	--					+				
Shopping Basket Pattern [Rossi99]			--		+	--	+	+							+
Transition Table [Weiss03]				+			+	+							
Continuation [Weiss03]				--		+	+			--		--			
Order Pattern [Fer00]			--	+	+	--						--			
Shipment Pattern [Fer00]			--	+	+	--						--			
Order/Shipment Pattern [Fer00]			--	+	+	--						--			
Shopping Process Pattern [Fer01]			--		+		+	+			+	+			
E-commerce Site [Welie03]		+	--		+		+	+			+	--	+		+

Table 5.7: Force Table for Process Patterns

Catalog Pattern: Provides a catalog class to collect products.

Context: Companies sell products of different types through an e-commerce application. These products are sometimes totally unrelated, such as books and food.

Problem: How does an application organize product information and improve the attraction of the web site?

Forces:

- A large variety of frequently changing products is sold on the web site.
- An application should provide an effective search and selection facility to attract users.
- General catalogs are created in an ad hoc manner and cannot be reused, thus it causes duplication and waste of time and effort.

Solution: A catalog class is defined to collect similar products together. The different aspects of products, such as detailed descriptions and related products, are separated into different classes, which include the Catalog class, Product class, SimilarProduct class, ProductInfo class, ProductObserver class and Customer class. ProductObserver uses the *Observer* pattern to look for changes and notify users of these changes.

Resulting Context:

- It provides an attractive way to display product descriptions.
- The more classes that are defined need, the more time and effort needed, which decrease the efficiency of the system.
- Mutable categories make the site hard to maintain.
- It is easy for users to find what they are interested in by catalog.
- This pattern is suitable for various kinds of web shops, so it is reusable.
- There is a drawback in that a product class representing a type of product can't represent an individual product, such as on a used-car site. Therefore, an additional class to indicate the product type is needed.

Figure 5.36: Force hierarchy for the *Catalog* pattern**Related Patterns:**

- To improve the performance in search, Search facilities, such as *Advanced Search* and *Selectable Search Space* are used together with the *Catalog* pattern.
- The *Catalog* pattern that is implemented by Stock Manager uses the *Observer* pattern to look for changes, and includes *Container/Context*.

Stock Manager: Uses class models to define the relationship of each component.

Context: For a good e-commerce web site, the site owner may want to keep track of quantity and location of items in stock, and update these quantities. The goal of the original author of this pattern was to help the inventory system manage production according to the different stages of manufacturing or production, from component ordering to product shipping. However, it is still useful for site owners to manage their products in stock.

Problem: How does the site owner keep track of their product information, including item type and locations?

Forces:

- The pattern must describe a fundamental semantic unit.
- Any modification of units should notify their dependences for example any transfer or use of items anywhere should update the inventory.

Solution: A strategy using class models defines the relationship between StockRoom, Distribution, Inventory, Product, Stock, ComponentInventory, ProductInventory and Component. The class diagram and sequence diagram are shown in [Fer00].

Resulting Context:

- Complex business logic needs extra effort in development and design.
- The product and component classes make it easy to write code to add new products, so that it improves scalability.
- The products are changed frequently, which makes code hard to maintain.

Related Pattern:

- The *Stock Manager* pattern implements the *Catalog* Pattern.

Shopping Basket Pattern: Provides a mechanism to keep track of user's selections and allow a transaction one time for all.

Context: An e-commerce web application makes shopping through Internet reality. Users navigate through the e-store to find what they intent to buy and then place an order. The impractical solutions are either to ask a user to buy each product once he finds it, or to force him to bookmark all the products and place an order in another session. In another word, this strategy requires one transaction for each product, which is not suitable to a case in which the user wants to buy dozens of different products.

Problem: How does an application simplify the "check-out" process so that it may enable users to select multiple products and check them out in one session?

Forces:

- It is impractical to require one transaction for one product.
- Users need to track the status of selection during navigation.
- The pattern should not interfere with the overall web site navigational structure.

Solution: This strategy provides users with a metaphorical shopping basket, allowing users to keep track of selected products. It is like a user shopping with a basket in a supermarket, and storing the goods in the basket during traversal. The user just needs one transaction at the end of shopping process for all the products he selected. This pattern is easy to implement by adding a button or link beside each product description so that the product is added into a list by clicking.

Resulting Context:

- An elaborate interface needs extra development effort.
- Users easily modify and delete selected items.
- A simple operation, that of clicking a button, is used to add product into the shopping basket, so that it increases the performance of customers and the usability of system.

- Complex calculation before checking out needs more computing time.
- Mechanisms such as list or session are used at the back end to keep selection status persistent.
- Users can review the basket any time during selection so that it is predictable for users.

Figure 5.37: Force hierarchy for the *Shopping Basket Pattern*

Examples: Figure 5.38 is a successful example of this pattern. It is possible for users to modify the quantity of each item by entering the desired number in “Qty” textbox, or to delete ones by clicking the “delete” button. The first line on the top of the page, “view cart” indicates the current step.

Figure 5.38: Example of Website using the *Shopping Basket/Cart* pattern

Related Patterns:

- Customers may need to keep track of the shopping selection, thus *Shopping Basket* is applied conjunctly with the *Dynamic Configuration* pattern and *List Builder* during *Shopping Process*.

Transition Table. A mechanism to define rules for the legal transitions between pages

Context: While designing an e-commerce application, the navigational design of the web site usually is described by storyboards, a site map, and some wireframes. Storyboards only describe rough sketches of user interaction. A site map documents the organization of the site, but it only shows the key relationships between pages. Wireframes document the important page elements such as search boxes, and the layout of specific pages [Wodtke03], but it doesn't describe the links between pages.

Problem: How do site designers specify the detailed navigation logic without getting lost in too many links and without forgetting important navigational paths?

Forces:

- The pattern should describe the relationships between pages and the navigation paths.
- The existing methods, such as storyboards, a site map and wireframes, are still ambiguous.

Solution: This strategy defines transition rules for the legal transitions between pages. A transition rule defines what actions corresponding to a specific page can be performed next. The left side of each transition rule is the name of the user action, and the right side contains the set of user actions that can be performed next. The side effects are also indicated by “?” denoting side-effect free query actions and “!” denoting update actions.

`<action> => <next action 1> | <next action 2> | ...`

Resulting Context:

- This pattern has negative effect on efficiency as the complex navigation logic may make design harder.
- The navigation logic described by transition rules can be used to derive test cases.
- This pattern helps to ensure whether the navigation logic is implemented consistently, and to predict the steps for developers.

- Transition rules can help to maintain and extend the code by being a reference.

Figure 5.39: Force Hierarchy for the *Transition Table* Pattern

Related Pattern:

- To keep the process consistent, *Continuation* relied on *Transition Table* is used to control *Shopping Process*.

Continuation. Passes context information for processing a user request by encoding each following action and its parameters within form parameters.

Context: When *Central Dispatcher* generates a page corresponding to a user action, this page needs all the information required to perform the next action (or a set actions) identified by *Transition Table*.

Problem: How does an application pass the information to generate the next page?

Forces:

- The solution should be simple.
- Sending context data unencrypted raises security and privacy risks.
- Relying only on *Continuation* may make applications brittle against interruptions.

Solution: Pass context information for processing a user request by encoding each new action and its parameters within form parameters such as URL parameters, hidden parameters and submit buttons. This can be implemented by encoding the context of an action in URL parameters, in a separate form, or one form with decorated input fields.

Resulting Context:

- Passing context information is easier than *Session*, but may violate security and privacy.
- The context information passed indicates the next step, which keeps the process consistent.
- Losing context or session expiration may interrupt the process so that the reliability is low.
- Client script is hard to test, which reduces maintainability.

Figure 5.40: Force hierarchy for the *Continuation* Pattern

Related Pattern:

- In a long transaction, customers may be interested in guaranteeing a persistent state so that *Continuation*, relying on *Transition Table*, is often employed together with *Session*, which stores context information on the server.

Order Pattern: Describes the order process by three classes: Customer class, Product class and Lineitem class.

Context: Ordering a product is a very common real-life problem. Users need to place an order for a certain kind of product or service from an e-commerce website.

Problem: How does an application describe a request for a product or service?

Forces:

- Users need to know the status of each step while shopping online.
- Users usually request a type of product instead of an individual product.
- The requests must be apprehended in an accurate way.
- The order is one of the processes of shopping that must be correlated with others, such as shipment and delivery.

Solution: The *Order* pattern describes the request by three classes: customer class delegating a customer who places the order, product class delegating a type of product, and lineitem class delegating the quantity of products. An order consists of a collection of line items describing a particular type of product and the quantity ordered. The class model diagram, with the state of the order and a sequential diagram, are shown in the original paper [Fer00] so they are omitted in this thesis.

Related Patterns:

- The *Order* pattern can't be used in isolation so that it comes with the *Shipment* pattern in a semantic operation.
- In processing checkout, we are interested in undertaking a desirable degree of scalability and efficiency hence *Value Object* is adopted in conjunction with *Order*.

Shipment Pattern: Describe the shipment process using shipment class, customer class and invoice class.

Context: After a user places an order on an e-commerce website, the product corresponding to the order should be delivered to the user.

Problem: How does an application describe the shipment of an ordered product or service?

Forces:

- The shipment should correspond to a respective order and to a receiver.
- The shipment should contain the product information and the amount to be paid (an invoice).
- How to deliver and pay for the product is beyond the scope of this pattern.

Solution: Three classes, shipment, customer and invoice, describe the shipment process that is applied to a customer with an invoice for a payment. Closing the shipment or the invoice is recorded in a log once the shipment is completed. The class model, state charts and sequential diagram are shown in the original paper [Fer00], which will not be repeated in this thesis.

Resulting Context: See *Order/Shipment* pattern.

Related Patterns:

- *Shipment* is the consequent process of *Order*.

Order/Shipment Pattern: Describes the shop processes including receiving, canceling and delivering an order by the combination of order and shipment patterns.

Context: Users place an order from an e-commerce website, and each order corresponds to a shipment. However, sometimes users may cancel the order.

Problem: How does an application describe the problems in real-life order and shipment processes?

Forces:

- The lifecycle of order and order fulfillment process needs to be optimized.
- The order fulfillment process needs to be maintained in case that a customer may change his/her minds.
- The detailed implementation of this pattern should not be included.
- The pattern must provide a fundamental semantic unit so that it is easy to apply to a variety of situations.

Solution: A combination of order pattern and shipment pattern describes a solution corresponding to the realization of the following use cases: receiving an order, canceling an order and delivering an order. In receiving an order scenario, a user's information, such as name and mailing address, is recorded, and their credit card is checked when the user places an order. A pre-requested order can be cancelled in canceling an order scenario. In the last scenario, an invoice is created once the shipment is checked against the order, and consequently the product and its corresponding invoice are delivered to the user according to the user's information provided in the first scenario.

Resulting Context:

- The complex relationship between object classes may need more design and development effort, accordingly it decreases the performance of the system due to computing time.

- It is easy for users to modify personal information so that it increases maintainability.
- Checking out numerous products in one transaction saves navigation time, thus it increases the performance for customers.

Figure 5.41: Force Hierarchy for the *Order/Shipment* Pattern

Related Patterns:

- The *Order/Shipment* pattern is the combination of the *Order* pattern and the *Shipment* pattern based on the semantic aspect of application model, which is used in *Shopping Process*.

Shopping Process Pattern: A strategy demonstrating the real-life shopping cart in supermarkets is provided.

Context: An e-commerce application makes customers shopping online as real as shopping in a physical store. The customer can look for the products that he is interested in, choose what he intends to buy, add them to a shopping cart, and check them out in the same session.

Problem: How does an application describe a shopping process in a precise way?

Forces:

- Customers may be disoriented through the steps of a shopping process, so the application should provide a clear and distinct indication of the process for customers.
- The steps of the process may need to be changed when new products or new business models are added.
- Several payment methods should be provided to satisfy the needs of more customers.
- The web application should be secure, otherwise customers will not trust and choose it.

Solution: The most common strategy is based on the concept of using a shopping cart in supermarkets. A user is allowed to have more than one shopping cart containing his selections. An order is placed when the user decides to check out his/her selections, and then an invoice is produced for each order. The class diagram is provided in the original paper [Fer01].

Resulting Context:

- An elaborate interface and complex functionality need extra development and design effort; consequently it decreases the efficiency.

- This pattern reduces the complexity of the shopping process. For example, the customer information needs not be reentered for member customers. Consequently, performance for customers is increased.
- The *Shopping Process* pattern allows each customer to have more than one shopping carts keeping the status persistent.
- The step that the current user is in is indicated by the *Explicit Process* pattern that is part of the *Shopping Process* Pattern.
- This pattern is independent of the type of products so that it can be applied to various web shops from selling food to those selling books. It is reusable.
- It is easy to add authorization to the exist structure. For instance, the access to the process can be restricted to specific roles, so it increases the security of system.

Figure 5.42: Force hierarchy for the *Shopping Process* pattern

Related Patterns:

- Sometimes customers may be interested in “picking up” products in traversal, therefore, *Dynamic Configuration* provides a convenient way to add the current item into a *Shopping Basket* by displaying an “add” button beside the products.

- The demand for process indication is fulfilled by undertaking the *Explicit Process* pattern during the shopping process.
- The requirement for keeping track of selections, as well as selection review is supported by using *Shopping Basket*.
- According to the semantic process, the *Order* pattern and the *Shipment* patterns are employed together to complete the shopping process.

E-Commerce Site: Provides a virtual store with basic functions for users to shop online.

Context: Users want to buy products or services online, so an application that makes users shopping online seem like reality is very necessary.

Problem: How does an application realize purchasing products online?

Forces:

- The basic model about online shopping helps the architecture to build an e-commerce website with basic functionalities.
- This basic model should be easy to expand.

Solution: The *e-commerce site* pattern provides a basic idea to create a “virtual store” where customers can browse, choose and pay for their selections in one go. A common e-commerce site includes three components: product database, customer profiles and shopping cart mechanism. Because the customers can’t touch the products before buying them, the web application makes users “feel good” about the products via animation or 3D models that are included in the *Product Configurator* pattern.

Resulting Context: The *E-commerce Site* pattern is composed of the previous patterns in this section. The force hierarchy representation for the *E-commerce Site* pattern is shown in Figure 5.43 and only includes the positive support. It indicates the non-functional requirements (efficiency and security) needed to be taken into account during further design.

Figure 5.43: Force hierarchy for the *E-commerce Site* pattern

Related Patterns:

- *E-commerce site* includes a shopping cart mechanism that includes the *Shopping Process* pattern, *Shopping basket*, *Order*, *Shipment*, and the *Order/Shipment* patterns.
- In the original document [Welie03], the author only includes the *Hotlist*, *Product Comparison*, *Login*, and *Shopping basket* patterns to demonstrate a basic model of an e-commerce site, but as a matter of fact all the patterns discussed in this thesis can be part of an e-commerce site.

5.4.2 Atomic Patterns

The *Value Object* pattern and the *Money* pattern are widely used atomic patterns. Money needs to be calculated while checking out at the end of shopping process. So they are discussed in this section.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistence	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
Value Object [Fowler02]			+	-		+	+						+		+
Money [Fowler02]			+	-		+	+						+		+

Table 5.8: Force Table for Atomic Patterns

Value Object: Defines a small simple object, like money or a date range, whose equality isn't based on identity.

Context: The money and date are the significant objects when customers check out of an e-commerce application. A reference object uses the identity within the programming system for equality, for instance, the built-in identity of object oriented programming language, or the primary key in a relational database. The key feature of the reference object is that it is passed around by reference, thus it may decrease the system performance. Usually it is used to represent large objects such as orders, or customers.

Problem: How does an application describe the small objects such as money and date?

Forces:

- The object should be small and easy to create.
- Changing the value in a value object may cause alias collision.
- Sometimes a pure object-oriented strategy may decrease system performance.

Solution: This strategy provides a simple object using value as the basis for equality. As value objects are small, they are often passed around by value instead of by reference. It is recommended that the value object be made immutable to avoid alias collision.

Resulting Context:

- Since a value object is easy to create, it saves time on development, which increases the efficiency.
- If the value is different, a different object needs to be created, so it would take more resources when numerous objects are created. Consequently, efficiency is decreased.
- The object is immutable, therefore it is hard to maintain but it increases persistency.

- The object is easy to pass around and independent of the platform so that it increases the scalability of the system.
- The object equalized by a value is small to be passed, thus it is easy to be created. The serialization of objects may improve the performance of the system.
- A simple object is easy to create, so usability is improved.

Figure 5.44: Force hierarchy for the *Value Object* pattern

Related Patterns:

- In processing checkout, we are interested in undertaking a desirable degree of scalability and efficiency hence *Value Object* is adopted in conjunction with *Order*.
- *Money* is a special case of *Value Object*.

Money: Represents a monetary value.

Context: Customers manipulate money to purchase products from an e-commerce application. Money is calculated in the check-out process by computers. There are still many problems about calculation. For example, multiple currencies exchange may cause currency differences to be lost. The other subtle problem is in rounding errors.

Problem: How does an application handle the previous problems?

Forces:

- The class should include the numeric amount and the currency.
- The money class should have its equality and hash code operations based on the currency and amount.
- The algorithm should avoid losing or gaining a penny.

Solution: Provides a money class with fields for the numeric amount and the currency, with some embedded methods. In order to avoid rounding errors, a solution that allows users to declare the rounding scheme when they call the method is recommended and another thing taken into account is to absolutely avoid the floating point data type in the amounts. The method `convertor()` is used to convert money from one currency to another, which allows users to specify rounding, looks up an exchange rate and multiplies by it. The method `allocator()` is provided to avoid losing or gaining pennies when allocating a sum of money to several accounts by subtracting from what you have allocated to the last allocation.

Related Patterns: *Money* is an instantiation of *Value Object*.

5.5 Server-side Patterns

The web shares information through HTTP and HTML. With the evolution of e-commerce applications, the web is rapidly being transformed into a data-exchanged and transaction-intensive tool. It not only shows information-oriented static pages, but also provides activity-intensive actions.

Suppose a customer enters an Internet store. He registers by filling in some forms, searches for desired products, puts them into a shopping cart, and then checks out. As we know, the HTTP is a stateless protocol. All the information may be lost at the end of process with only HTTP. How does an application keep the state persistent in the whole purchase process? The following table includes most patterns on this issue.

Note: The *Client Session State* pattern is used on client side, which is out of the scope of this section. But it is also collected in this table in order to distinguish the different usage of client session from server sessions.

	Accuracy	Attractability	Efficiency	Maintainability	Performance (Customer)	Performance (System)	Persistency	Predictability	Readability	Reliability	Reusability	Security	Scalability	Trust	Usability
Session [Yoder97]			--	--		+	+						--		
Unique Id [Weiss03]			--			--	+					+			
Client Session State [Fowler02]			+			+	+			--		--			
Server Session State [Fowler02]			--	--		--	+			+		+			
Server-side Session [Man01]			--	--		--	+			+		+			
Property [Foote98]			--			--							+		
Serialized LOB [Fowler02]			+	--		--	+		+						
Database Session State [Fowler02]			--			--	+			+		+			
Application Server [Man01]			--			--				+		--	+		

Table 5.9: Force Table for Server-side Patterns

Session. The *Session* pattern provides a session object that holds all variables shared by many objects.

Context: Users navigate and purchase products through an e-commerce application. The application needs to keep track of the product information and the current user's information throughout the whole shopping process until the user checks out.

Problem: How does an application keep track of shared information throughout the system?

Forces:

- Global variables could reduce development effort and would be easy to maintain.
- The shared information could change frequently.
- Each object may share some values.
- Passing objects in the application makes development more complicated and lowers system performance.

Solution: A session object, which holds the important variables that need to be shared, is recommended. This session object, which provides a common interface for all components to access, is passed around. Certain user information can be encapsulated in the project, such as a user's role and privileges. A session should be terminated after a certain amount of inactivity.

Resulting Context:

- Building session objects may need extra time and effort.
- Passing a session object instead of passing multiple variables improves the performance.
- State is kept persistent by session objects.
- All variables placed in a single session object are hard to maintain and split. Consequently, the scalability is reduced.

Related Patterns:

- The requirement to define which process or subsystem is allowed for whom is supported by applying *Roles* passed by *Session*.
- To keep the selection state persistent in the *Shopping Process*, *Session* object is used jointly with *Easy Undo and Explicit Process*, as well as *Continuation*, which is applied by the *Server Session State* pattern.
- To improve security, the *Unique Id* pattern is adopted with *Session* identification.

Unique ID. Provides a unique session id using cryptographic hash.

Context: When you use *Session* to keep the state of user information persistent in an e-commerce application, the unencrypted session Id is easy to be guessed.

Problem: How does an application protect session Id from being stolen?

Forces:

- Unencrypted session Id may raise the security risk.
- Session Id should be abandoned when the session expires.

Solution: A unique session Id is provided by using a cryptographic hash. To validate the session id, the hash should include some identification of sources, such as source address, source port, and user agent.

Resulting Context:

- The cryptographic hash keeps the session Id secure.
- Session Id is used to keep consistent status for a user.
- The complex encrypting and decrypting process may diminish system efficiency.
- The process of encrypting and decrypting may consume more computing time, therefore performance is reduced.

Figure 5.45: Force hierarchy for the *Unique Id t* pattern

Related Pattern:

- To satisfy the requirement for keeping the session state persistent and to preserve desirable security, *Session*, passed by *Serialized LOB* is exploited jointly with *Database Session State*, and *Session* is identified by *Unique Id*, which is stored in the Database as a key.

Client Session State. Stores session state on the client.

Context: Session is used to keep information persistent in an e-commerce application, as HTTP is a stateless protocol. Although the session can be stored in the server, for small amounts of data, the time cost in transferring data to the server may decrease the system performance. Another issue is that the data will be lost with server failure.

Problem: How does an application keep the session data for small amounts of data?

Forces:

- Storing data in the server may consume time in transferring so that it decreases performance.
- Server session state is lost when the server fails.
- A server side session needs an identifier on the client side.

Solution: *Client Session State* provides a strategy that stores session state on the client side. The common implementations are: URL parameters, hidden fields, and cookies. URL parameters are easy to implement, but there is a limit in the size of an URL. Hidden fields are not secure since anyone can obtain the data by looking at the page source. Cookies can be limited and turned off by the client. However, the solution that involves detecting whether cookies are enabled, and if not, then using URL rewriting, may be a good choice. Usually, the developers have to use *Client Session State* for session identification. That would be better be one number, which won't burden the page and URL. In order to avoid session stealing, most platforms work with a random session ID. For the security issue, encryption and decryption may reduce the risk, but they may bring a performance burden.

Resulting Context:

- Client session state is easy to implement, which increases efficiency.
- As hidden fields are easy to obtain, it is not secure. Encryption and decryption can reduce the security issue but cause a performance burden.

- Storing data at the client side instead of transferring the server session increases performance.
- The goal of this pattern is to keep a user's information persistent. Thus, it increases system persistency.
- Since the user can turn off cookies, the system reliability is lower.

Figure 5.46: Force hierarchy for the *Client Session State* pattern

Related Patterns:

- To improve reliability in the event of client failure, *Client Session State* is always used together with *Server Session State* in the Shopping Process.
- In this case, the session ID of *Server Session State* is always retrieved by *Client Session State*.

Server Session State. Keeps the session state on a server system in a serialized form.

Context: Client session state may lose all information when the client machine crashes. It is only suitable for small amounts of data.

Problem: How does an application store the session state for large amounts of data or objects?

Forces:

- Client session state only fits for small amounts of data and does not guarantee data persistence.
- Large amounts of data stored in a client machine may cause a space burden.
- *Server Session State* will also lose data if the server fails.

Solution: *Server Session State* provides an approach that keeps the session state on a server system (usually on the application server) in a serialized form. The form is usually as simple as possible. Several platforms provide a simple binary serialization mechanism but it is unreadable for human. Another form is text (popularly as an XML file). To support efficient clustering or failover, the server session state is stored on an accessible storage, such as on a shared server, however it may take longer time to activate the server. Therefore, the serialized server session state is stored in the database using a session table indexed by the session ID. This table would have a *Serialized LOB* to hold the serialized server session state. In this case, the performance may be dependent on the database. For example, the IBM's WebSphere can serialize a session bean into a BLOB in DB2 to allow multiple application servers to obtain its state.

Resulting Context:

- Simple programming may increase the efficiency of the system, however the complicated design issues such as version upgrading may decrease efficiency.
- Persistent storage keeps the session state persistent under clustering and failover.
- Serializing objects need extra time, which may decrease the performance.

- The binary form is not readable for a human, and it is hard to upgrade because a serialized object is needed to read it back.
- Using multiple shared servers (such as state servers) may keep state persistent in the event of the failure of one server.
- The data is stored on an application server itself, either in a file system or database. That is not accessible directly from the client, thus it increases the security.

Figure 5.47: Force hierarchy for the *Server Session State* pattern

Related Patterns:

- To improve reliability in the event of client failure, *Client Session State* is always used together with *Server Session State* in the Shopping Process.
- In this case, the session ID of *Server Session State* is always retrieved by *Client Session State*.

Server-side Session. Uses a session object on the server.

Context: The web becomes a transaction-intensive environment in an e-commerce application. In consideration of customer registration, product selection, and transaction, the application needs to track the state of the shopping process by each request since HTTP is a stateless protocol.

Problem: How does an application keep the state of information related to a user's interaction?

Forces:

- The state information needs to be held until the customer leave the site or the current transaction is over.
- Clients are not allowed to access the state information due to the security of the system.
- Server side session may fail when the server crashes.
- The server may become a bottleneck since it must be able to handle concurrent access by a large amount of users.
- It is impractical to store large data on a client such as small-memory PDAs.

Solution: This approach stores a session object on the server as well as on the client. The client stores additional information such as ID for the server-side session that copes with the server fail-over. For instance, as the client-side cookie stores a key to the location of the *Application Server*, the client can find all information by switching to the secondary server when the primary server fails. The liability is that changing the state of the session object may be very complicated.

Related Patterns:

- *Server-Side Session* is a kind of *Server Session State*.
- Typically, the *Server-Side Session* using the *Property* pattern acts like a keyed dictionary, and often employed together with *Application Server*.

Property. A mechanism for accessing, altering, adding, and removing properties or attributes at runtime.

Context: In the context of applying *Server-Side Session* to pass the session object in an e-commerce application, the object may need to be added in some fields or properties, such as inventory information.

Problem: How does an application augment the state of an individual object at runtime?

Forces:

- The session objects may be changed in certain cases.
- The code to modify the properties of objects that are passed around the network might be complex.

Solution: This strategy provides a mechanism to access, alter, add and remove properties or attributes at runtime. The operations such as `addProperty()`, `removeProperty()`, `hasProperty()`, `setProperty()`, and `getProperty()` are presented to allow developers to deal with the properties. In some implementations, the property is added by `setProperty()` or deleted by `Property.Remove`. Most implementations use dictionaries or hashtables. When properties are rare, you can store a mapping between instances and their property list elsewhere to avoid the overhead of providing an additional field to store a list object. Obviously, this is a time-consuming approach as a second dictionary looks for space.

Related Patterns:

- *Server-side Session* uses *Property* to modify the state of session object.

Database Session State. Stores session data as committed data in the database.

Context: Usually, the data on each request needs to be recorded in durable storage.

However, the *Server Session State* may fail with the application server's failure.

Problem: How does an application keep persistent data when *Server Session State* fails?

Forces:

- This pattern should support clustering and failover with *Server Session State* in your particular application server.
- Caching the server object will burden the resources.
- Pulling data from the database and writing it back will cost time.
- The data needs to be stored in a durable storage.

Solution: *Database Session State* provides an approach to store session data as committed data in the database. The session state is always a mix of session data that's only local to the current interaction and committed data that's relevant to all interactions. The key issue resolved in this pattern is to separate such session data. The one strategy is to add a field to each database row that may have session data, such as storing a session ID as a pending field, which is easy to find all data for a particular session. Another solution is a separate set of pending tables that are exact clones of the real tables. In order to get data rollback, you can copy all the data that may be modified into pending tables. You can modify it there, and commit it back to the record tables at the end of the session. This can be done only if session ID is a pending field as part of the key.

Resulting Context: As the force hierarchy shows (Figure 5.48), using *Database Session State* improves reliability, security and persistency, but trades off the cost of design and time performance.

Figure 5.48: Force hierarchy for the *Database Session State* pattern

Related Patterns:

- To promote security, the separate pending tables may need *Serialized LOB* to hold the serialized *Server Session State*.

Serialized LOB. Saves a graph of objects by serializing them into a single large object (LOB), which is stored in a database field.

Context: Object models such as customer, product, and shipment, are used in the shopping process design. However, the information of some object models often contains the complicated relationship between small objects. For example, an order object may include the information of the customer who makes an order, the information of the product that the particular customer is buying, and the shipment information including who will deliver and the address of customer. It is hard to keep all these objects persistent as table rows related to each other.

Problem: How does an application manipulate such objects?

Forces:

- It is complicated to store a whole graph of objects in the table rows.
- A serialized large object may take more space in the database.

Solution: This strategy stores the serialized objects in a database field. There are two forms of serialization: binary (BLOB) and textual characters (CLOB). The BLOB is simple to implement and uses minimum space, but it requires that the database supports a binary data type. The CLOB needs more space and a specific parser for textual format. In this case, XML is a common route for the standardization. In order to avoid duplicating data, you need to assure the serialized LOB can't be reached from anywhere but a single object.

Resulting Context:

- Serialized LOB is simple to implement.
- A BLOB takes minimum space, while a CLOB needs more space.
- A BLOB is not readable for human, but a CLOB is readable.
- Serialization of objects keeps the state persistent.

- If you modify the serialized object, it is impossible to read all its previous serializations so that it is hard to maintain.
- Serializing objects may need extra time.

Figure 5.49: Force hierarchy for the *Serialized LOB* pattern

Related Patterns:

- High security is obtained by applying *Serialized LOB* that holds serialized *Server Session State* in a database.

Many customers may access an e-commerce application at the same time. In consideration of keeping track of information for respective customer, the published patterns on session are documented in this section.

A *Session* object sharing the global information is passed around to objects that need any of its values, such as a user's role and privileges. However, it is difficult for web application specialists to add session objects late in the development process, and it might be tedious to store many values in a session object. Because the *Client Session State* pattern only fits for small amounts of data, and is easy to be hacked into, the *Server Session State* pattern and the *Database Session State* pattern provide more secure and reliable approaches. The *serialized LOB* pattern stored in the database is transferred to the client by session ID that is captured by *Client Session State*. The *Server-side Session* is a kind of *Server Session State*, which might not survive under network damage or a client crash. *Database Session State* stores the committed session data in the database. It provides the most durable approach since the pending table may hold the previous copy of data when the network fails or the client crashes.

Each pattern has a different contribution to the system. The combination of them may reach more soft-goals, which is shown in Figure 5.48 as follows. *Server side sessions* increase reliability and security, but client session increases performance and efficiency for easy application.

Figure 5.50: Force hierarchy for patterns on sessions

Application Server. A program that handles all application operations between users and backend business applications or database.

Context: The architecture of an e-commerce application may involve web browsers, web server and the Internet. The web browser displays the data to users and gathers data from HTML pages in the role of thin clients. The web servers cope with HTML content and send it back to the browsers, as well as receiving user data from them. The Internet connects the clients to the server through HTTP.

Problem: Where is the application put?

Forces:

- The interaction between users and the application is only through web browsers.
- All application data to users and user data to the applications pass through the web server.
- Time and money should be taken into account while deploying the system.

Solution: The *Application Server* pattern provides a strategy that separates the web server from the application, and the web server plays a *Single Access Point* for a user's application interaction. The web server doesn't deal with the data, and just sends requests to the application that handles business logic but not the visual aspect of the presentation.

Resulting Context:

According to Figure 5.51, force hierarchy for *Application Server*, the scalability, security and maintainability of the system have been improved; however, it trades off efficiency in development and time performance.

Figure 5.51: Force hierarchy for the *Application Server* pattern

Examples: *Application Server* plays an important role in an e-commerce architecture, which provides an extremely scalable and secure transaction engine for dynamic e-commerce application. Figure 5.52 shows how the *Application Server* Pattern works in J2EE infrastructure.

Figure 5.52: Sample of *Application Server* using in J2EE

The web browser sends a form request to a web server through HTTP. The web server will not deal with the data, but sends a request to the application server. The web server works like content delivery. All the business transactions will be handled on the application server. In J2EE entity beans implement data exchange with the database at the backend. After the application server receives data, it returns the content to the web server instead of presenting it directly. The web server uses server pages, such as ASP, JSP or PHP, to generate HTML, which is presented on the browser. How does the web server access the application data? Another pattern *Singleton* [GoF95] provides a global point of access to it. If the system needs load balance, a naming service can be considered, such as JNDI. Nevertheless, there is still an issue to be taken into account that a bottleneck might be made when malicious users access the server, which may present a single failure.

Related Pattern:

- In order to keep state persistent, session patterns such as *Server Session State*, *Server-Side Session* and *Database Session State* are explored.
- Customers may be interested in high reliability and security in transaction, hence *Application Server* is adopted in conjunction with *Serialized LOB*.

5.6 Chapter Summary

In this chapter, we have documented most patterns used in an e-commerce domain from client side to server side. In each section, according to different tasks, the patterns are grouped in a table to compare the contributions to the system. Some patterns are published using pattern form, but some are not. We have extended some existing patterns and re-written some patterns using a new proposed pattern representation with force hierarchy, which interprets forces as non-functional requirements, and uses the NFR framework to analyze the context of the applicability of these patterns and the consequence of employing them.

6 Case Study: A Main Feature of A Web Store

As part of this thesis, a case study about analyzing and designing a main feature of a web store was examined by applying the approach proposed in the previous chapter.

A web store design involves the interaction of several features tightly joined, which includes product introduction, customers' profiles, new release recommendations, search facilities, a purchase process, a billing system, product delivery and security. As the purchase process is the main function of an e-commerce application, it is studied here as a motivating example to illustrate how a set of patterns previously described in separate papers can be effectively used together. As a result, it allows us to devise a pattern language in terms of interrelationships between them and the associated non-functional requirements.

6.1 Selecting the Patterns

Before buying products, customers need to look after the desired ones and then carry out a purchase process pursuing order and shipment.

Based on the interrelationship of e-commerce patterns in Figure 5.1, we focus on *Shopping Process* firstly, and then map to its related patterns by tracing the directly connected links. Thereby, there are eight alternative patterns (*Client Session State*, *Server Session State*, *Order/Shipment Pattern*, *Session*, *Shopping Basket*, *Transition Table*, *Catalog Pattern* and *Explicit Process*).

To obtain desired products, search facilities related to the *Catalog Pattern* need to be considered, which are *Advanced Search* and *Selectable Search Space*.

Since *Shopping Basket* is an essential mechanism for the purchase process, the relevant patterns (the *Dynamic Configuration Pattern* and the *List Builder* pattern) linking to it need to be taken into account as well.

Hence, there are totally twelve patterns around the *Shopping Process* pattern, which can be considered in addressing several challenges involved in designing the main feature of a web store. The process of selecting is shown in the Figure 5.2.

Figure 6.1 Selecting patterns using pattern roadmap

6.2 Analyzing the Issues in the Purchase Process

For the search facilities, two non-functional requirements (or quality attributes) are considered as having crucial importance at the design level: Accuracy of search results and time Performance. Whereas for the purchase process, the other three non-functional requirements, Maintainability, Performance in customer interaction and Persistency, are regarded as essential influence at this high level design stage. The new representation of patterns with explicit forces and non-functional tables help us precisely catch the negative and positive supports, on which the forces are employed in the process of selecting patterns. Since a set of related patterns are grouped in a force table, we can easily get the contributions and side-effect of the pattern when comparing each force column.

The design issues are progressively addressed in terms of semantic adoption as follows:

- The requirements for retrieving accurate search results and ending in a satisfactory degree of performance are fulfilled by exploring *Advanced Search* jointly with *Catalog* and *Selectable Search Space*.
- The need for keeping selected products persistent during the whole shopping process is addressed by *Session*, *Client Session State* and *Server Session State*. Although the use of *Server Session State* implies giving up on Performance, the combination of *Session* and *Client Session State* fills this gap.
- The demand for providing customers an explicit process and showing the current shopping status is achieved by adopting *Shopping Process*, *Transition Table*, *Explicit Process* and *Shopping Basket*. The search patterns trade-off on

maintainability due to frequent changes in products updating, however system maintainability can be promoted by using *Transition Table*.

- In the shopping process, we need to provide a convenient way for customers to select products and store them in a basket. This is supported by applying the *Dynamic Configuration Pattern* and the *List Builder* pattern.

According to the interrelationship graph (see Figure 5.1) and the non-functional requirements tables, we have obtained the structure of the main feature of a web store that arises from increasingly selecting the considered patterns while the design issues are analyzed and the expected non-functional requirements are satisfied. The force hierarchy of the selected patterns is shown in the following Figure 6.2.

In terms of the way to select the appropriate patterns shown in this case study, we can summarize the steps for designing an e-commerce application as follows:

1. With the requirements in mind, select the possible patterns as alternatives using pattern roadmap.
2. Analyze the design issues and non-functional requirements.
3. Compare the related patterns using force table to determine which patterns are exactly satisfying the non-functional requirements.
4. Repeat step 2 and 3, and obtain all desired patterns from the alternatives.

Figure 6.2: Force hierarchy for some patterns of e-store

6.3 Chapter Summary

The case study contained in this chapter serves to illustrate how the new pattern representation and the interrelationships in the e-commerce pattern graph help designers to decide what patterns can be applied, and when, during a design process.

As shown in the case study, the *Catalog Pattern* [Fer01], the *Transition Table* pattern [Weiss03] and the *Shopping Process Pattern* [Fer01] are originally documented in different papers, nevertheless they incorporate with each other to satisfy the same soft goals (non-functional requirements). The interrelationships in the e-commerce patterns graph (see Figure 5.1) has established a basic pattern map, which helps designers make more informed decisions on pattern selection.

An explicit representation of the non-functional requirements behind pattern interrelationships is a key aspect in deriving a pattern language [Araujo03]. A conceptual framework for designing a reusable application has been developed by increasingly refining the pattern interrelationships in a specific domain and adding more related patterns in accordance with non-functional requirements.

7 Conclusions

The goal of this thesis is to identify a document of links between most of the existing patterns for e-commerce applications as found in the literature in terms of a proposed pattern classification, and an extended pattern representation that uses the NFR framework. For each step of the process of building an e-commerce application, from working on client side aspects to working on server side aspects, the contribution of each pattern in terms of its effect on non-functional and functional requirements has been discussed. The extended pattern representation helps the system designer select a pattern during the process of building an e-commerce application.

In this chapter, we summarize how well we met the expected objectives, and the contributions of the thesis. We will also give an outlook on directions for future work.

7.1 Review of the Objectives

By systematically reviewing the previously documented pattern representations, Alexandrian form, software design pattern forms, and the extended pattern representation, a new pattern representation has been introduced in this thesis. Our new pattern representation outlines the forces by using explicit lists, and adds auxiliary force hierarchy in the “resulting context” part, which is based on Araujo’s master’s thesis.

In accordance with the proposed pattern representation, the candidate patterns that have originally been documented in separate papers using inconsistent representations have been rewritten in a consistent manner, while also establishing previously undocumented relationships between the patterns.

In examining the existing pattern classifications, a new classification of candidate patterns in the e-commerce domain has been introduced in terms of the prevalent e-commerce architecture, which are presentation tier, workflow tier, business logic tier and data tier.

We used a case study to design the main features of an online store as an example to show how the proposed facilities allow a designer to make more informed decisions on selecting patterns in light of functional and non-functional requirements. It also provides an example of the applicability of our pattern compilation.

7.2 Summary of the Contributions

In this section we review how well the thesis achieves the claimed contributions.

- This thesis provides an index of most existing patterns in the e-commerce domain. Through the support of our pattern index in chapter 4, which gives a short description of each pattern, pattern users may quickly locate a particular pattern.
- According to the prevalent n-tier architecture of e-commerce systems, a new classification of e-commerce patterns has been created. The patterns in the index table are sorted by four tiers. Therefore, the designer may quickly select the patterns through the tier they might be used on. This forms a guide to help designers select patterns in designing an e-commerce application.
- The use of force hierarchies improves the understandability of the patterns. As shown in the chapter five, force hierarchies are used in our new pattern representation. All candidate patterns have been rewritten in the new pattern representation in a consistent manner. As a result, the pattern representation supports the understanding of pattern contributions and trade-offs.
- The links in the pattern map identify the interrelationships between the patterns, which may support more new pattern languages in the e-commerce domain.

- The pattern map with the links between the candidate patterns provides an overview of the patterns and their associations. Resulting from the case study, this establishes a means to assist pattern selection by narrowing down the number of candidates. As a consequence, this helps designers to select the appropriate patterns for building an e-commerce system.
- A new approach, comparing patterns using force hierarchies, has been supplied to document the contributions made by a set of patterns. This serves to facilitate a pattern user's decision on the most appropriate patterns by allowing them to compare the contributions and side effects explicitly emphasized in the force hierarchy.

7.3 Future Work

The proposed approach is clearly applicable to pattern selection as the conducted case study showed. The future directions include:

- Integrate other kinds of patterns, which are not included in this thesis, to build a framework. It would support its suitability in pattern selection under a different context and provide the opportunity for pattern refinement.
- Build an implementation-based case study as a form of validation or benchmark by integrating these patterns. It would allow a pattern user to observe how useful different patterns would be, or allow a user to observe which patterns are actually used in the system.
- Build a tool for generating force hierarchies and selecting patterns automatically. This tool would receive the non-functional requirements as input and generate force hierarchies and patterns as output. It would help pattern users easily obtain the appropriate patterns according to system requirements.

Appendix: The index of patterns for e-commerce application.

The following index is ordered by pattern name alphabetically.

Pattern Name	Type	Page	Brief Description
Advising [Rossi00]	N	114	Helps customers choose what they want by user profiles, bestseller, products on sale and subsystems. Known as: Product Advisor
Advanced Search [Welie03]	N	143	Offer a special advanced search function with extended term matching, scoping and output options.
Application Server [Man01]	S	202	A program that handles all application operations between users and backend business applications or databases
Behavioral Anticipation [Gar97]	UI	60	Indicates the effect and consequence of behavior.
Behavioral Grouping [Gar97]	UI	62	A solution that groups control objects on interface by functionality.
Check Point [Yoder97]	UI	46	Provide a mechanism for security check and validation.
Carrot and a Stick [Per99]	UI	65	A mechanism that withholds sensitive content until users submit more information.
Central Dispatcher [Weiss03]	UI	71	Handle all requests in a single “entry” script that dispatches them to the appropriate handler.
Customizable window [Welie03]	N	130	Provides a window where selected components can be adapted or clicked away.
Catalog Pattern [Fer01]	I	159	A catalog class to be a product collection.
Continuation [Weiss03]	I	167	Passes context information for processing a user request by encoding each next action and its parameters within form parameters.
Client Session State [Fowler02]	I	189	Stored session state on the client
Dynamic Configuration Pattern [Lya98]	N	125	Provides a means to perform a selection over a set of options from web.
Database Session State [Fowler02]	S	196	Stores session data as committed data in the database.

Easy Undo [Rossi00]	N	104	Returns any specific state of the process instead of backtracking.
Explicit Process [Rossi00]	N	106	Provides a feedback about purchase process through “progress” line or enumerating steps.
E-commerce Site [Welie03]	I	177	Provides a virtual store with basic functions for user to choose and pay for goods online.
Footer Bar [Welie03]	UI	56	The navigation links of whole site and condition of use are shown on footer bar.
Front Controller [Fowler02]	UI	69	A controller that handles all requests for a web site.
Frequently Asked Questions (FAQ) [Welie03]	UI	88	Provide an organized section on frequently asked questions and short answers.
Hotlist [Welie03]	N	121	Show a list of most popular items.
Login [Welie03]	UI	42	Ask user to login by email address and password combination if necessary.
Language Selector [Welie03]	UI	81	List all available language in its own language at the prominent position.
List Builder [Welie03]	UI	91	Present the total list and provide editing functionality on selected items.
Money [Fowler02]	I	182	Represents a monetary value.
My Site [Welie03]	N	128	A solution to create user’s own web contents. Related Patterns: Customization Window
News Pattern [Lya98]	N	115	Shows new release information on part of pages to capture the attention of customers. Known as: News Box [Welie03]
Newsletter [Welie03]	N	119	Send users a newsletter about updated content regularly.
Offload Work to Client [Weiss03]	UI	54	Offload some functions to client-side scripts and leaves some validation at the server as well.
Opportunistic Linking [Rossi00]	N	112	Provides more related links about new products to capture attention.
Order Pattern [Fer00]	I	169	A class that requests for a product or service
Order/Shipment Pattern [Fer00]	I	171	Describes the placement and fulfillment of an order.
Premium Content Lock [Welie03]	UI	55	A visual marking used to lock premium content.
Policy Statement [Per99]	UI	48	A policy statement of how the information users provided will be used.

Page Controller [Fowler02]	UI	73	An object that handles a request for a specific page or action on a Web site.
Paging [Welie03]	UI	90	Present the results grouped in pages.
Product Configurator [Welie03]	UI	96	Use a direct and visual version of the configured product for users.
Product Comparison [Welie03]	UI	98	Show a matrix of different products and features.
Plan B [Per99]	UI	100	ALT tag instead of web graphics is used to allow more users to perceive web content.
Process Feed-Back [Gar97]	N	107	Provides a feedback about operation status.
Push Communication [Rossi00]	N	123	Provides a push-based communication between user and web store using subscribing, mailing and personalizing site, etc.
Roles [Yoder97]	UI	49	Define the different permission and access rights to different group of users.
Registration [Welie03]	UI	52	Offer users a registration after first use of the data to avoid reenter the same information.
Required Field Markers [Per99]	UI	58	Provides a special notation scheme in required fields.
Property [Foote98]	S	195	A mechanism to access, alter, add, and remove properties or attributes at runtime
Single Access Point [Yoder97]	UI	44	Set up only one way to log into the system.
Separate Content From Presentation [Weiss03]	UI	77	Separate presentation from application logic.
Simple Search Interface [Lya99]	N	136	A simple search interface with one field that satisfies a large proportion of users. Known as: Simple Search[Welie03]
Simple Search [Welie03]	N	136	Offer a search functionality consisting of a search label, a keyword and a filter.
Selectable Keywords [Lya99]	N	138	Provides a list of possible keywords according to previous search result.
Structured Answer [Lya99]	N	140	A brief abstract of target page and a taxonomy information space compose of structured answer.
Selectable Search Space [Lya99]	N	145	Provides select category (sub-space) to search.
Search Tips [Welie03]	N	148	Offer help on keywords and on matching options.

Selectable Search Engine [Lya99]	N	150	Add link to an existing search engine.
Search Area [Welie03]	N	153	Combine different kinds of search functionality on a dedicated area.
Stock Manager [Fer00b]	I	161	A mechanism that keeps track of stocks and their locations.
Shopping Basket Pattern [Rossi99]	I	162	A mechanism that keeps track of user's selection in navigation by a shop cart. Known as: Shopping Cart [Welie03]
Shipment Pattern [Fer00]	I	170	Describes the shipment of an ordered product or delivery of a request service.
Shopping Process Pattern [Fer01]	I	173	A process that provides shopping cart, order, checkout and invoice.
Server Session State [Fowler02]	S	191	Keeps the session state on a server system in a serialized form.
Server-side Session [Man01]	S	194	Session object at server side holds state information.
Serialized LOB [Fowler02]	S	198	Saves a graph of objects by serializing them into a single large object (LOB) stored in a database field.
Session [Yoder97]	S	185	A session object holds all variables shared by any objects.
Transform View [Fowler02]	UI	75	A view that processes domain data element by element and transforms it into HTML.
Template View [Fowler02]	UI	79	Renders information into HTML by embedding markers in an HTML page.
To the Top [Welie03]	UI	87	Provides a link to the top of the page at position of the list.
Tabbing [Welie03]	UI	93	Divide the data in sections and show one section at a time.
Transition Table [Weiss03]	I	165	A mechanism to define rules for the legal transitions between pages
Unique Id [Weiss03]	S	187	Provides a unique session id using cryptographic hash.
Value Object [Fowler02]	I	180	An object whose conceptual identity is based on a combination of values of its properties.
Wizard [Welie03]	UI	95	Step users through the checkout process.
What They See is All They Get [Per99]	UI	63	Puts the most important information always on visible space.

Bibliography:

- [Alexa64] Alexander, C.: *“Notes on the Synthesis of Form”*. Harvard University Press, Cambridge, MA. 1964
- [Alexa77] Alexander, C.: *“A Pattern Language”*. Oxford University Press, New York 1977.
- [Alexa79] Alexander, C.: *“The Timeless Way of Building”*, Oxford University Press, 1979.
- [Araujo02] Araujo, I.: *“A Design Process Based on Patterns and Non-Functional Requirements”*, Master’s Thesis, School of Computer Science, Carleton University, 2002
- [Araujo02A] Araujo, I., and Weiss, M.: *“Linking Patterns and Non-Functional Requirements”*. Pattern Languages of Programming (PLoP), 2002
- [Araujo03] Araujo, I.: *“Making Explicit the Non-Functional Requirements in Pattern Languages”*, ChiliPLoP, 2003
- [Alan03] Alan, C.C.: *“Unifying pattern catalogs (towards a pattern for pattern lexicons)”*, SugarloafPLoP, 2003
- [Chung00] Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: *“Non-functional Requirements in Software Engineering”*, Kluwer Academic Publishers, 2000
- [Carlson01] Carlson, D.: *“Modeling XML applications with UML: practical e-business applications”*, Addison Wesley, 2001
- [Coa97] Coad, P.: *“Object models: Strategies, patterns, and applications”* (2nd Edition), Yourdon Press, 1997
- [CDT03] Berman, J.: *“Web Site Privacy Policies”*, <http://www.cdt.org/privacy/guide/start/privpolicy.html>, as viewed on December, 2003
- [Coplien95A] Coplien, J.O.: *“A Development Process Generative Pattern Language”*, Bell Laboratories: <http://www1.bell-labs.com/user/cope/Patterns/Process/index.html>
- [Coplien95B] Coplien, J. O., Schmidt, D.C., *“Pattern Languages of Program Design”*, Addison Wesley, ISBN: 0-201-60734-4, 1995
- [Coplien96] Coplien, J. O., *Software patterns*. SIGS Books & Multimedia, 1996

[EuroPLoP] European Pattern Languages of Programs:
<http://www.hillside.net/patterns/EuroPLoP/>

[King96] King, M.: “*Evaluating natural language processing systems*”, ACM Press, 1996

[Fer00] Fernandez, E.B., Yuan, X., and Brey, S. : “*An analysis pattern for the order and shipment of a product*”, PLoP 2000

[Fer01] Fernandez, E.B., Liu, Y., and Pan, R.Y.: “*Patterns for Internet Shops*”, PLoP 2001

[Fer00b] Fernandez, E.B.: “*Stock Manager: An analysis pattern for inventories*”, PLoP 2000

[Foote98] Foote B., Yoder J., “*Metadata and Active Object-Models*”, PLoP 1998

[Fowler02] Fowler, M. “*Patterns of Enterprise Application Architecture*”, Addison Wesley, 2002

[Fer00] Fernandez E. B., “*Stock Manager: An Analysis Pattern for Inventories*”, PLoP 2000

[Gar97] Garrido, A., Rossi, G., and Schwabe, D. : “*Pattern Systems for Hypermedia*”, Procs. PLoP 1997

[GaTech98] Kehoe, C., Pitkow, J., and Rogers, J.: “*GVU’s Ninth WWW User Survey Report*”, Office of Technology Licensing, Georgia Tech Research Corp., Atlanta, 1998
http://www.cc.gatech.edu/gvu/user_surveys/papers/

[Gamma93] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., “*Design patterns: Abstraction and reuse in object-oriented designs.*” Springer-Verlag, 1993.

[GoF95] Gamma E., Helm R., Johnson R., Vlissides J. “*Design Patterns. Elements of Reusable Object Oriented Software.*”, Addison Wesley, 1995

[Gross01] Gross, D., Yu, E.: “*From Non-Functional Requirements to Design through Patterns*”, Faculty of Information Studies, University of Toronto, Toronto, Ontario, Canada, 2001

[Hyatt03] Hyatt, K.: "N-Tier Application Development with Microsoft.NET", last accessed February 2003,
<http://www.microsoft.com/belux/nl/msdn/community/columns/hyatt/ntier1.msp>

[KoalaPloP] KoalaPloPTM is one of the series of conferences in the PLoP:
<http://www.cs.rmit.edu.au/KoalaPloP/>

[Lynch03] Lynch, M., "*E-commerce and the Challenge of Trust*",
<http://www.ml.com/woml/forum/ecommerce1.htm>, April 2003.

[Lya99] Lyardet, F., Rossi, G., and Schwabe, D.: "*Patterns for adding search capabilities to web information systems*", EuroPloP 1999, <http://www-di.inf.puc-rio.br/~schwabe/papers/Europlop99.pdf>

[Lya98] Lyardet, F., and Rossi, G.: "*Patterns for dynamic websites*", PLoP 1998,
<http://jerry.cs.uiuc.edu/~plop/>

[Lynch97] Lynch, P., Horton, S., Yale C/AIM Web Style Guide
<http://info.med.yale.edu/caim/manual/index.html>, 1997.

[Man01] Manolescu, D.A., and Kunzle, A.E.: "*Several Patterns for eBusiness Applications*", PLoP 2001

[McPhail00] McPhail, C. J.: "*Deciding On A Pattern: Towards A Distributed Decision Aid that Supports the Selection of an Object-Oriented Software Pattern*", a thesis of master of science information and system science, school of computer science, Carleton University, Ottawa, 2000

[Ong03] Ong, H. Y., and Weiss, M., "*Rewriting a Pattern Language to Make it More Expressive*", Expressiveness of Pattern Languages, ChiliPloP, 2003

[Buschmann96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: "*Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*", John Wiley & Sons, Ltd., (1996).

[PloP] PLoP conference pages: <http://st-www.cs.uiuc.edu/~plop>

[Per99] Perzel, K., Kane, D : "*Usability Patterns for Applications on the World Wide Web*", SRA International, PLoP 1999

[Pfleeger87] Pfleeger, S. L.: "*Software Engineering: The Production of Quality Software*", ISBN:0-02-395720-4, 1987

[Rossi99] Rossi, G., Schwabe, D., and Lyardet, F.: “*Patterns for designing navigable information spaces*”. Pattern Languages of Programs IV, Addison Wesley, 1999.
<http://www-di.inf.puc-rio.br/~schwabe/papers/PLoP98.pdf>

[Rossi00] Rossi, G., Lyardet, F., and Schwabe, D.: “*Patterns for E-commerce applications*”, Procs. EuroPLoP’2000 <http://www.coldewey.com/europlop2000/>

[Rum91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W.: “*Object-Oriented Modeling and Design*”. Prentice Hall, Englewood Cliffs, NJ, 1991

[Rising00A] Rising, L.: “*The Pattern Almanac 2000*”, Addison Wesley, 2000, ISBN 0-201-61567-3

[Rising00B] Rising, L.: “*Design Patterns in Communications Software*”, SIGS REFERENCE LIBRARY, 2000, ISBN: 0521790409

[Rossi01] Rossi, G., Schwabe, D., Danculovic, J., and Miaton, L.: “*Patterns for personalized web applications*”, Procs. EuroPLoP’2001,
<http://www.hillside.net/patterns/EuroPLoP/>

[Smith98] Smith, J., and Jones, P., Special edition on Patterns, CACM, May 1998
<http://www.cs.wustl.edu/~schmidt/CACM-editorial.html>

[Thayer90] Thayer, R., Dorfman M.: “*System and Software Requirements Engineering*”. IEEE Computer Society Press, 1990

[Wodtke03] Wodtke, C., “*Information Architecture: Blueprints for the web*”, New Riders, 2003

[Weiss03] Weiss, M., “*Patterns for Web Applications*”, Pattern Languages of Programming, PLoP 2003

[Welie03] Welie, M., “*Interaction Design Patterns*”, www.welie.com/patterns/index.html, last accessed October 2003

[Yoder97] Yoder, J., Barcalow, J.: “*Architectural Patterns for Enabling Application Security*”, PLoP 1997

[Yeh82] Yeh, R.: “*Requirements Analysis – A Management Perspective*”, Proceedings of COMPSAC’82, 410-416