INFORMATION TO USERS

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

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

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

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

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

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

UMI[®]

A Proto-Pattern Language for Human-Computer Interaction

By

Minoo Azad, M.Arch

**A thesis submitted to
the Faculty of Graduate Studies and Research
In partial fulfillment of the requirements for the degree of
Master of Architecture**

**Carleton University
Ottawa, Ontario
April 20, 2000
© 2000 Minoo Azad**

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

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

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

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

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

0-612-52376-4

Canada

The undersigned recommended to
The Faculty of Graduate Studies and Research
acceptance of the thesis

"A Proto Pattern Language for Human-Computer Interaction"

submitted by:
Minoo Azad, Msc.
In partial fulfillment of the requirements for
the degree of
Master of Architecture
(Architecture and New Technology)

Chair:

Greg Andonian

Head of Department:

Benjamin Gianni

Thesis supervisors:

Dr. Dwight Deugo

Benjamin Gianni

.....

External Examiner:

Francis Bordeleau

.....

Carleton University
Date: 12 May 2000

A proto-pattern Language for Human-Computer Interaction

I wish to extend my special thanks to Dr. Dwight Deugo, Professor of the School of Computer Science, who acted as my thesis supervisor, for his advice, guidance, support, and encouragement during the course of this thesis research work. Also, I owe him my awareness of the application of the pattern theory in the software design.

I also wish to thank Professor Ben Gianni from the School of Architecture (who acted as my thesis co-supervisor), Dr. Vincent Mosco from the School of Journalism and Communication, Dr. Alex Ramirez from the School of Business, the Institute of Inter-Disciplinary Studies, the Writing Tutorial Services of the School of Linguistics and Applied Language Studies, Mrs. Anne Paxaver, and Mrs. Sandra Estabrooks for their guidance, support, and help.

Minoo Akad

March 2000

Carleton University

Ottawa

A Proto Pattern Language for Human - Computer Interaction

Abstract

Human-computer interaction explores the relation between human (user) and system (computer) in the context of a socio-technical environment. The roots of the complexity and diversity of this field are deep in its social nature (human factor) as well as the changeability of the growing new technology.

We need a shared language for communicating insights and experiences to resolve common difficult problems encountered in this field. One way to create a shared language is to build a pattern language. A language of this sort is a collection of patterns which share similar assumptions, terminology, and context.

The goal of my thesis is to write such a pattern language for human-computer interaction. To organize the space of an informative artifact and the structure of my proposed proto-pattern language, I borrowed the theory of representation and spatial organization of the pattern based architecture. The idea of creative user participation is also extracted from this kind of architecture.

Christopher Alexander's pattern theory helped me to find a proper representation for this idea to correspond with the contemporary way of thinking about the system. To write this language, I have used the vocabulary, terminology, and conventions of the software pattern community.

My proto-pattern language advocates the application of participatory design instead of conventional joint application design approach. This proto-pattern language is primarily concerned with the communicative aspects of human-computer interaction and does not cover all subjects of this field.

A Proto Pattern Language for Human-Computer Interaction

List of Figures and Tables

Chapter 1: Morphology of the Concept

Chapter 2: Current Challenges in the Field of Human-Computer Interaction

Chapter 3: Standards and/or Pattern

Chapter 4: A Proto-Pattern Language for Human-Computer Interaction

Chapter 5: Conclusion

Appendices

Notes, References, Glossary

A Proto Pattern Language for Human-Computer Interaction

Table of Contents

Figures and Tables	09
List of Figures	
List of Tables	
Chapter 1:	14
Morphology of the Concept	
Chapter 2:	27
Current Challenges in the Field of Human-Computer Interaction	
What is Human-Computer Interaction?	
Why is Human-Computer Interaction important?	
Technologies	
Intellectual Foundation	
Future Strategical Themes	
Future Technological Trends	
Current Challenges	
Conclusion	
Chapter 3:	37
Standard and/or Pattern	
What is a Standard?	
What is a Software Pattern?	
Pattern Definitions:	
In the Dictionary	
In Mathematics	
In Architecture	
In Software	
What is a Pattern Language?	
What are Proto-Patterns?	
Patterns as a Solution	
Standards and/or Patterns	
Chapter 4:	56
A Proto-Pattern Language for Human-Computer Interaction	
Section1: Background	57

	4
Part 1: Concepts	58
I will borrow some basic ideas and concepts from:	
Pattern Based Architecture	
Alexander's Pattern Theory	
Software's Pattern Community	
Human-Computer Interaction's Pattern Community	
Part 2: Approach	65
Architectural Approach	
Christopher Alexander's Approach	
What are Self-Organizing Systems?	
Software Approach	
Conclusion	
Part 3: Format	74
Different Formats	
Choosing a Format	
Part 4: Components of a Pattern	79
Name	
Problem	
Context	
Forces	
Solution	
Rationale	
Resulting Context	
Known Uses	
Related patterns	
Sketch	
Examples	
Part 5: Forces	86
System Collaborations	
User Collaborations	
Conclusion	
Part 6: Sketch	94
What is a Sketch	

Sketch in Software Design**Part 7: Conclusion** 101**Section 2:****A Proto-Pattern Language for Human-Computer Interaction** 105

Introduction

Proto-Pattern Language Summary

How to Use these Patterns

Conventions

Problem/Solution Summary

Section 3: Patterns 113**Part 1: Spatial Patterns** 114

Spatial Model Pattern

Space Pattern

Part 2: Organizational Patterns 126

Formation Pattern

Scalability Pattern

Grading Pattern

Navigation Pattern

Sequence Pattern

Short-Cut Pattern

Transition Pattern

Default Values of an informative artifact

State Pattern

Delay Pattern

Non-Sequential Pattern

Part 3: Delimiter Patterns 209

Window(Shape) Pattern

Panel(Surface) Pattern

Part 4: Generic Forms Patterns 218

Conceptual Widgets Pattern

Concrete Widgets Pattern

	6
Part 5: Social Patterns	227
User Participation Pattern	
Chapter 5:	236
Conclusion	
Achievements	
Future works	
Appendices	246
Appendix1: Patterns in Architecture	247
Design Paradigm	
Fundamental Concepts of the "Pattern Based Architecture"	
Appendix2: Christopher Alexander's Pattern Theory	253
Section 1: Biography and Publications	254
Biography	
Publications	
Section 2: Alexander and Evolution of an Idea	260
Notes On The Synthesis of Form	
The Oregon Experiment	
A Pattern Language: Towns, Buildings, Construction	
The Timeless Way of Building	
A New Theory of Urban Design	
A Foreshadowing of 21st Century Art: The Color and	
Geometry of Very Early Turkish Carpets	
The Language of the Carpets	
Domestic Architecture	
The Nature of Order	
Section 3: Evaluation of Alexander's ideas	299
Appendix3: Patterns in Software	306
Software Patterns	
Historical Process	
Appendix4: Patterns in Human-Computer Interaction	313

Background
Abstracts

Appendix5: Related Patterns	327
Section 1: Organization and Complexity	328
Section 2: Frameworks and Patterns	333
What is a FrameWork?	
The Advantages of Using Frameworks	
Frameworks and Class Libraries	
Frameworks and Patterns	
Section 3: State and its Patterns	339
State Pattern(Change an Object's Class at Run-Time)	
Example	
Section 4: Window's Patterns	345
Definitions	
Window, Where it Starts	
Task Window Pattern	
Technological Background	
Model-View-Controller	
Model-View-Controller as an Aggregate Design Pattern	
Model-View-Controller with Presentation Model	
Visual Proxy Pattern	
Section 5: Panel and its Patterns	355
Definitions	
Dividing window into Few Panes	
Section 6: Widgets and Patterns	358
Common Ground A Pattern Language for Human-	
Computer Interaction	
Iconic Reference Pattern	
Section 7: Joint Application Development and Participatory Design	378
What is Joint Application Development?	

Phases of Joint Application Development
 Advantages of Joint Application Development
 What is Participatory Design?
 Motivations of Participatory Design
 Goals of Participatory Design
 Advantages of Participatory Design
 Comparing Participatory Design and Joint Application Development
 Combination of the two methodologies
 Conclusion

Notes

398

Chapter 1: Morphology of the Concept
 Chapter 2: Current Challenges in the field of Human-Computer Interaction
 Chapter 3: Standard and/or Pattern
 Chapter 4: A Proto-Pattern Language for Human-Computer Interaction
 Chapter 5: Epilog
 Appendices:
 Appendix1: Patterns in Architecture
 Appendix2: Christopher Alexander's Pattern Theory
 Appendix3: Patterns in Software
 Appendix4: Patterns in Human-Computer Interaction
 Appendix5: Related Patterns

References

421

Glossary

446

*A Proto Pattern Language for Human-Computer Interaction***List of Figures and Tables**

List of Figures

Figure 01: The scope of my proposed proto-pattern language.

Figure 02: Time-lines of three main stream technologies in the field of human - computer interaction.

Figure 03: Interrelationships among the multiple topics that are involved in the field of human - computer interaction.

Figure 04: Waterfall model of application development in software design.

Figure 05: Spiral model of application development in software design.

Figure 06: Fractal model of application development in software design.

Figure 07: Alexandrian-Format for writing a pattern.

Figure 08: Portland Format for writing a pattern.

Figure 09: Modern Alexandrian Format for writing a pattern.

Figure 10: Canonical Format for writing a pattern.

Figure 11: Gang of Four Format for writing a pattern.

Figure 12: Pattern-Oriented Software Architecture Format for writing a pattern.

Figure 13: The format that will be used for writing patterns of my proposed proto-pattern language.

Figure 14: A farm.

(Sketch in architecture; A set of composable diagrams by Alexander)

Figure 15: A water distribution system.

(Sketch in architecture; A set of composable diagrams by Alexander)

Figure 16: A water collection unit.

(Sketch in architecture; A set of composable diagrams by Alexander)

Figure 17: A village.

(Sketch in architecture; A set of composable diagrams by Alexander)

Figure 18: Half-Object + Protocol.

(Sketch in software; A set of diagrams by Coplien)

Figure 19: Half objects are asymmetric.

(Sketch in software; A set of diagrams by Coplien)

Figure 20: Adding more Half Calls to the Call.

(Sketch in software; A set of diagrams by Coplien)

Figure 21: The structure of my proto-pattern language.

Figure 22: Different possible configurations for organizing the content of an informative artifact.

Figure 23: A linear sequence of informative nodes has no need to use a complex graded system.

Figure 24: A graphic representation of a graded Informative artifact.

Figure 25: A graphic representation of the main path of navigation in a graded Informative artifact.

Figure 26: Options for managing a sequence.

Figure 27: A graphic representation of the short-cut path in a graded informative artifact.

Figure 28: A profile of the short-cut path in an informative artifact with a graded formation.

Figure 29: A graphic representation of transitions between different nodes in an informative artifact with a graded formation.

Figure 30: A graphic representation of non-sequential formations of an informative artifact.

Figure 31: Two examples of the pattern based architectural space.

Figure 32: Alexander by Max Kisman.

Figure 33: The master plan of the University of Oregon.

Figure 34: The language of carpets: words (part 1) & (part 2).

Figure 35: The overall landscape of a pattern language (User Interface Software Pattern.)

Figure 36: Benefits and costs of using frameworks.

Figure 37: Framework and Class libraries.

Figure 38: Framework and Design patterns.

Figure 39: A Workspace for editing an address entry in the email handler.

Figure 40: A Workspace object when it is in the saved state.

Figure 41: Model-View-Controller relationship.

Figure 42: Model-View-Controller class hierarchy.

Figure 43: Model-View-Controller with Presentation Model.

Figure 44: The Visual Proxy and its layers.

Figure 45: Joint Application Development, Phase 1: Project Definition.

Figure 46: Joint Application Development, Phase 2: Research

Figure 47: Joint Application Development, Phase 3: Preparation

Figure 48: Joint Application Development, Phase 4: The Session

Figure 49: Joint Application Development, Phase 5: The Final Document

Figure 50: A guide to Participatory Design practices.

Figure 51: Movement from attractor and pedestrian movement in a town center.

Figure 52: The hierarchical and horizontal relationships in an informative artifact with a graded formation.

List of Tables

Table 01: The process of development of patterns in my proposed proto-pattern language.

Table 02: The structure of the content of my thesis.

Table 03: Sets of goals and constraints in the development of an informative artifact that follows a user participation policy.

Table 04: Spatial Patterns.

Table 05: Organizational Patterns.

Table 06: Delimiter Patterns.

Table 07: Generic Forms Patterns.

Table 08: Social Patterns.

Table 09: Active Grade Transitions in an informative graded artifact.

Table 10: List of the published patterns by Plop and EuroPlop that are related to the field of human-Computer Interaction.

Table 11: Problem/Solution Summery of "A Pattern Language for Human-Computer Interaction Design" by Jenifer Tidwell (Part 1 to Part 9).

Table 12: Problem/Solution Summery of Iconic Reference Pattern.

Table 13: The summery of contrasts between Joint Application Development approach and Participatory Design approach.

A Proto Pattern Language for Human-Computer Interaction

Chapter 1:

Morphology of the Concept

*A Proto Pattern Language for Human-Computer Interaction***Chapter 1:****Morphology of the Concept**

At the beginning of the summer 1998, when I started thinking about the subject of my thesis, I found many documents and course offerings on the Internet related to the connection between human-computer interface issues and Critical Social Theory.¹ There are not anymore of them on the Internet today.

As I knew, Romanticism is a trend in modern culture, which exalts subjectivity against objectivity, life against rationality, concreteness against abstraction. Certainly the antagonism of Marx² toward the oppressive formalism of capitalist social life, analyzed and condemned in parallel critiques of "alienation" and "reification," is to some degree a tributary of that trend. And yet it would grossly distort the theories of alienation and reification to reduce them to a romantic protest against reason as is frequently suggested by contemporary critics.

In relation to technology, Critical Social Theory argues that technology is not a thing in the ordinary sense of the term, but an "ambivalent" process of development suspended between different possibilities. This "ambivalence" of technology is distinguished from neutrality by the role it attributes to social values

in the design, not merely the use, of technical systems. In this view, technology is not a destiny but a scene of struggle.

Critical Social Theory is about **values**, not about the system or design. The documents about human-computer interaction and user interface design that I have mentioned at the beginning referred to Critical Social Theory as a philosophical background.

The Institute of Interdisciplinary Studies ³ at Carleton University assisted me in locating professors with research interests related to my topic. I checked the basis of this idea with professor Vincent Mosco ⁴ from the school of Journalism and Communication. ⁵ His brief and straightforward discussions in September of 1998 cleared the sky for me. I was more or less on the right track. I understood that I had to refer to the social sciences to understand the basis of this debate and escape from the trap. The book "*Social Science, Technical Systems and Cooperative Work, Beyond the Great Divide*" [Bowker, et al., 97] provided a final insight. In this book Bowker, Star, and Turner have brightly explored the relation between social science and the new technology. By reference to this document I concluded that:

The attempt to understand how human and technical issues come together in Computing systems is now widespread but its inception as an area of academic

research only goes back to the 1960s. Initially, such research was informed by concerns about automation,⁶ as well as by psychological and management studies of user interface design and efficiency. It also included a healthy dose of social criticism and philosophically informed debate about the nature of thought and decision-making. The Scandinavian tradition of participatory design of computing which is called "co-determination" or "co-design," began in the late 1960s and was conducted jointly by social scientists, computing scientists, unions and workers. Researchers from this approach have studied workplaces in an ethnographic style, using the results to fuel the development of easier-to-use systems, which seek to enhance the workplace, not impoverish it.⁷

The developing socio-technical management approach, during the 1970's was an attempt by system designers and social scientists to cover the negative entropy inside the systems resulting from the reaction of users to the full coverage of systems. Different disciplines such as urban planning and political and technical systems have implemented versions of this approach based on their special needs during the 70's and later.

Computer-human interaction issues are an implementation of socio-technical management approach. The user centric design approach is the reaction of the system to the human factor. Any system design looks to be user centric if we look

at it from a system's viewpoint.

I have omitted all the work that I had done to clear this issue from my thesis. I preferred to refer to the more technical and professional literature. I tried to review definitions, hot-topics, previous achievements and future trends of the human-computer interaction as a scientific field. The result of this research is summarized in Chapter 2: Current Challenges in the Field of Human-Computer Interaction. This research showed me that there are some main points on which people who are working in this field agree. Most of them believe that human-computer interaction is a young field and the boundaries of its activities and responsibilities are not very clear and well defined. It has a multi disciplinary nature, and suffers from the lack of a shared literature to facilitate the communication between the members of its professional community.

This conclusion urged me to refer to the discipline of software design from which human-computer interaction is originated. This discipline has experienced the problem of a lack of the shared literature, since the early days of the development of the computer applications. One of the solutions that software design has found to this problem is blossoming in the software patterns.⁸ In the next step I tried to find out what the pattern and pattern languages are, where the idea came from, and how they could be applied in the field of computer science. I

discovered that I could refer to four main sources: Pattern Based Architecture, Christopher Alexander's Pattern Theory, Software Patterns and Human-Computer Interaction Patterns. The results of this research are included in Chapter 3: Standards and Patterns. This chapter includes the multiple definition of pattern (a solution to a problem in a given context), pattern language (a set of related patterns that are linked to each other through an organizational path), and proto-pattern language (a pattern language that is waiting to be tested in the real world). This chapter is supported by Appendix1 to Appendix4.

The goal of my work was clear: ***I should write a proto-pattern language for human-computer interaction.***

Could this language be different from those, written by software designers involved in this field? James Coplien's work "Space: The Final Frontier" in C++ Report, [Coplien, 98-1] referred me to Brenda Laurel's book "Computer as Theater" [Laurel, 93] and her idea about creating a theatrical metaphor for computer applications. The first question that the reading of this book brought to my mind was: "Theater is not interactive, how is its space a metaphor for computer applications?" I could remember that during the 70's many people who were involved in theater, cinematography and other related industries such as T.V. tried to find a way to facilitate the intervention of users in their products. Picasso⁹ is known for his efforts to popularize painting through mass production

of the artistic artifacts. These people created and supported the idea that art is inclusive not exclusive.

To be "inclusive" in art means that everybody has right to create a piece of artistic work, no matter if it has an artistic value. This concept is different from the possibility of the user participation in creating and changing the space of an artifact (a city, a table, or an informative artifact). All of those efforts for facilitating the intervention of the users in artistic products were unsuccessful due to the static nature of these media in relation to the user intervention or as it is called in urban planning "user participation." This phrase became the key for finalizing the goal of my thesis. The only spatial model that could work as a metaphor for computer applications should be extracted from the space of architecture and/or its twin urban design and Planning. This idea comes up from these facts: architecture is multipurpose (we live, work, enjoy in architectural space), it is a spatial system and architectural space has organization (just like all computer applications). Architecture has a synthetic nature (like all object-oriented computer applications), and it accomplishes a pure user participation that most intelligent computer application designers may have a desire for. Architecture has a shared vocabulary, it may be pattern oriented and it has a history of working as a reference discipline for computer science. To be pattern oriented became another key concept for my thesis. Not every kind of

architecture uses patterns. To be pattern oriented belongs to that realm of architecture that works on the synthesis of forms. In North American vocabulary this kind of architecture is called *Institutional Architecture*¹⁰ and is a twin of urban planning and design. At this point I summarized the goal of my thesis as follows:

To develop a proto- pattern language for human - computer interaction that uses pattern based architectural space as its abstract spatial model.

Next, I determined the scope of my proto-pattern language, as shown in Figure 1.

Figure 1 shows that from a technical point of view (software design) there is a mediator class that should work between model (that talks to the machine), and interface (that talks to the user). When a programmer writes a program at the API level (Application Programmer Interface) he is sitting on the user side (as a person) talking to the machine. At the interaction domain the same situation occurs at a higher level. Interaction designers talk to the program and the user. Therefore, there should be another mediator class at the interaction domain. Additionally the user wants to intervene in the application (customization and participation). Now it is clear that there are two domains over API in which two

Figure 1: The scope of my proposed proto-pattern language

different agents (software and interaction designers, or interaction designers and end users) are going to intervene. For this reason the patterns that are written in this field have a technical (when they are talking about a problem at under GUI level) or a social nature (when they are relating to the over GUI level). The presence of multiple agents in these domains is a reason for the complexity of the field of human-computer interaction. Unifying the directions of the efforts of these agents will decrease the complexity of this field to the benefit of the user.

Finally the goal of my thesis and its scope were determined, as follows:

To develop a proto-pattern language for human-computer interaction that uses pattern based architectural space as its abstract spatial model and which takes into consideration both technical and social aspects of an informative artifact.

To implement this goal, I have used the results of my research on four main sources: Pattern Based Architecture, Christopher Alexander's Pattern Theory, Software Patterns, and Human-Computer Interaction Patterns. I extracted the basic concepts, a methodology for developing my proto-pattern language and a format for writing the language and its patterns. All of these considerations are included in the first three parts of Section 1 (Background) of **Chapter 4: A Proto-Pattern Language for Human-Computer Interaction**. The next three parts of this section represent popular definitions for all components of the selected format and the reason for the modifications that I have applied to this format. The resulting criteria for developing my proto-pattern language are summarized in the last part of the Section 1 of Chapter 4.

My proto-pattern language and its patterns are documented in the second and third sections of Chapter 4 (A Proto-Pattern Language for Human-Computer Interaction and Patterns). This language has 18 patterns that are categorized in

five groups. Group 1 covers those patterns that are related to the abstract space modeling for the informative artifact. The next three groups include those patterns that realize this abstract model into a concrete feature. The last group is devoted to the user participation pattern. This section (Patterns) is supported by **Appendix 5 (Related Patterns)**, where the technological background of some of the patterns is introduced. The notes, references and a glossary of terms are attached to the end.

My proto pattern language is more concentrated on the communicative aspects of an informative artifact than on technical views. It does not cover all aspects of Human-Computer Interaction. As I have explained and illustrated in Figure 1, the scope of this field is very broad. This field covers many social and technical aspects. Writing a comprehensive pattern language for this field would require the time and efforts of many people.

The process of development of the patterns of my proto-pattern language is shown in Table1. Table 1 shows that at the first pass I could recognize fifteen patterns. By writing first versions of these patterns, I found out that there are two more patterns that I should add to the existing set of patterns to complete the structure of my proto-pattern language. Also, the first pass at developing patterns showed the necessity of dividing an existing pattern (Generic Forms Pattern) to two other patterns (Conceptual and Physical Widgets Patterns). Therefore, the

No	Pass 1	Pass 2 (Pattern Mining)	Versions							
			1	2	3	4	5	6	7	8
Spatial Patterns										
1	Spatial model Pattern		■	■	■	■	■	■	■	
2	Spatial Pattern		■	■	■	■	■	■	■	
Organizational Patterns										
3		Formation Pattern	■	■	■	■	■	■		
4	Scalability Pattern		■	■	■	■	■	■	■	
5	Grading Pattern		■	■	■	■	■	■	■	
6	Navigation Pattern		■	■	■	■	■	■	■	
7	Sequence Pattern		■	■	■	■	■	■	■	
8	Short-Cut Pattern		■	■	■	■	■	■	■	
9	Transition Pattern		■	■	■	■	■	■	■	■
10	Default Values of an informative artifact		■	■	■	■	■	■	■	
11	State Pattern		■	■	■	■	■	■	■	
12	Delay Pattern		■	■	■	■	■	■	■	
13		Non - Sequential Pattern	■	■	■	■	■	■	■	
Delimiter Patterns										
14	Window (Shape) Pattern		■	■	■	■	■	■	■	
15	Panel (Surface) Pattern		■	■	■	■	■	■	■	
Generic Forms Patterns (Widgets)										
16		Conceptual Widgets Pattern	■	■	■	■	■	■	■	
17		Physical Widgets Pattern	■	■	■	■	■	■	■	
Social Patterns										
18	User Participation Pattern		■	■	■	■	■	■	■	

Table 1: The process of development of patterns in my proposed proto-pattern language

Legend:

■	Previous Versions
■	Final Version

total number of patterns reached eighteen. According to Table 1, any pattern has three to seven versions prior to its final version. This experience showed me that the process of rewriting a pattern mutually elaborates the pattern and its related patterns to reach their final versions. The last eight columns of Table 1 represent

the number of versions of each pattern. Only the last (final) versions of the patterns are included in my thesis.

Appendices are used as supporting documents to give a brief information about some related topics. Some of these appendices help a computer scientist to understand the content of my thesis. The others contain the information that an architect needs to understand the content of my thesis. I included them as appendices intentionally to focus on the most related topics to the goal of my thesis and simultaneously keep its interdisciplinary nature on track.

To summarize Chapter 1 to 5 respectively represent subject-scope- process, problem definition, solution, implementation, and achievements of my thesis. Appendix 1 Introduces architectural patterns to software designers. Appendix 2 Introduces Alexander and his Pattern Theory to both disciplines and evaluates this theory. Appendix 3 Introduces software pattern movement to architects. Appendix 4 Introduces fifteen pattern languages and patterns for HCI to both disciplines. Part 1 of Appendix 5 Introduces the concept of organizational complexity, Part 2 of Appendix 5 Introduces the relations between pattern-framework-application and Part 3 to 6 of this appendix introduce some existing pattern and pattern languages that are used in my proto-pattern language to Architects. Part 7 of Appendix 5 introduces Joint Application Design and Participatory Design to both disciplines and compares these approaches.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 2:**

Current Challenges

in the Field of Human - Computer Interaction

What is Human-Computer Interaction?

Why is Human-Computer Interaction Important?

Technologies

Intellectual Foundations

Future Strategic Themes

Future Technological Trends

Current Challenges

Conclusion

*A Proto Pattern Language for Human-Computer Interaction***Chapter 2:**

Current Challenges in the Field of Human-Computer Interaction

This chapter introduces achievements, future trends, and current problems in the field of human-computer interaction. ¹¹

What is Human-Computer Interaction?

" Human-computer interaction is the study of how **people** design, implement, and use interactive computer systems, and how **computers** affect individuals, organizations, and society. " [Myers, et al., 96]

The Special Interest Group on Computer-Human Interaction (SIGCHI) of the Association for Computing Machinery (ACM) has defined this field as a discipline as follows:

" Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them." [ACM, 92]

Human-computer interaction encompasses not only ease of use but also new interaction techniques for supporting user tasks, providing better access to information, and creating more powerful forms of communication. It involves input

and output devices and the interaction techniques that use them, how information is presented and requested; how the computer's actions are controlled and monitored; all forms of help, documentation, and training; the tools used to design, build, test, and evaluate user interfaces; and the processes that developers follow when creating interfaces.

Why is Human - Computer Interaction Important?

The importance of interaction between human and computer results from different factors such as: user expectations, economic considerations, crucial considerations, and effectiveness. Users expect highly effective and easy-to-learn interfaces and developers now realize the crucial role that interface plays. [Myers, et al., 92] Employing the processes, techniques, and tools developed by the Human - Computer Interaction designers can decrease costs and increase productivity. ¹² There are well-known catastrophes that have resulted from not paying enough attention to the human-computer interface. ¹³ Effective user interfaces to complex applications are indispensable. The recognition of their importance in other disciplines is increasing. Therefore, an interdisciplinary collaboration is needed to fully address the many challenging research problems in the field of human-computer interaction.

Technologies

Research in the field of human-computer interaction started as early as

1960,¹⁴ as shown in Figure 2. It has been successful at fundamentally changing the world of computing, and has resulted in the growth of the four main streams including: Direct Manipulation of Graphical Objects, Windows, Hypertext, User Interface Management Systems (UIMSs), and Toolkits. There are many other techniques, including work that led to the development of drawing programs, paint programs, animation systems, text editing, spreadsheets, multimedia, 3D, virtual reality, interface builders, event-driven architectures, usability engineering, and many other significant achievements.

Figure 2: Time-lines of three mainstream technologies in the field of human-computer interaction¹⁵

Intellectual Foundations

The intellectual foundations of human-computer interaction derive from a

variety of fields: computer science, cognitive psychology, social psychology, perceptual psychology, linguistics, artificial intelligence, anthropology, and the development of object-oriented programming. This style of programming comes from early work on Simula but was further developed and refined in Smalltalk as a natural way to implement user interfaces. [Myers, et al., 96]

Figure 3 is a representation of the interrelationships among the multiple topics that are involved in the field of human-computer interaction.

Figure3: Interrelationships among the multiple topics that are involved in the field of Human-Computer Interaction ¹⁶

Future Strategic Themes

The future strategic themes in the field of human-computer interaction are predicted as follows: [Myers, et al., 96]

1- Universal Access to Large and Complex Distributed Information:

The "global information infrastructure" expands at unprecedented rates, and as a result, the potential user community of databases and other information systems is becoming startlingly large and rather non-technical. From this point of view interfaces with minimal technical sophistication, Information-access interfaces, and Information-discovery interfaces are considered as some of the most strategic themes for the future researches in the field of human-computer interaction.

2- Education and Life-Long Learning: Computationally assisted access to the information has important implications for education and learning as evident in current discussions of "co-laboratories" and "virtual universities." The development and evaluation of new educational technologies such as multimedia systems, interactive simulations, and computer-assisted instructional materials are part of the future mandate of the field of human-computer interaction.

3- Electronic Commerce: Another important theme revolves around the increasing role of computation in our economic life and highlights central to

human-computer interaction issues that go beyond usability to concerns with privacy, security, and trust.

4- End-User Programming: End-user programming will be increasingly important in the future. No matter how successful interface designers are, systems will still need to be customized to the needs of particular users. Although there will likely be generic structures, such systems and agents will always need to be tailored to meet personal requirements. The use of various scripting languages to meet such needs is widespread, but better interfaces and understandings of end-user programming are needed.

5- Information Visualization: This area focuses on graphical mechanisms designed to show the structure of information and improve the cost structure of access. The importance of information visualization will increase as people have access to larger and more diverse sources of information such as digital libraries and large databases, which are becoming universally available through the Web. Visualizing the WWW itself and other communication networks is also an important aim of information visualization systems. ¹⁷

6- Computer-Mediated Communication: Computer-mediated communication includes tools ranging from email to those involved in newer forms of communication via computers (real-time video and audio

interactions). Computer Supported Cooperative Work (CSCW) confronts complex issues associated with integration of several technologies, which have particularly difficult interface development challenges. This topic also demands a lot of work on the multi-person activities, and issues of security, privacy, and trust.

Future Technological Trends:

There are some general technological trends in the field of human-computer interaction, including connectivity (since the networked world becomes a reality), compatibility (since applications increasingly need to run across different platforms and code begins to move over networks as easily as data), and coordination (since the need to support multi person and organization activities is recognized). A few instances of these general trends include: computational devices and ubiquitous computing; speed, size, and bandwidth; speech, handwriting, natural language, and other modalities; 3D and virtual reality.

The Special Interest Group on Computer-Human Interaction of the Association for Computing Machinery has identified ten future trends in the field of human-computer interaction including ubiquitous communication, high functionality systems, mass availability of computer graphics, mixed media, high-bandwidth interaction, large and thin displays, embedded computation, group interfaces, user tailor-ability, information utilities. [ACM, 92]

As predicted by this group, through these developments the computing systems will appear partially to dissolve into the environment and become much more intimately associated with their users' activities. This is similar to the development of motion power. Once, strikingly visible, large, centralized water wheels were used to drive applications via belt drives, now electric motors are invisibly integrated into applications from VCRs to refrigerators.

Current Challenges:

Human-computer interaction is a complex and diverse field that explores the relation between human (user) and system (computer) in the context of a socio-technical ¹⁸ environment. The roots of the complexity and diversity of this field are deep in its social nature (dealing with human factor) as well as in the changeability of the growing new technology.

An important factor in driving the diversification of human-computer interaction is customization. As systems become increasingly customizable due to technologies such as component software, more and more design (in the sense of front end creation, application programming, and software configuration) is being done in-house. This may mean that traditional management information systems departments are playing a role, or external consultants are involved, or end users themselves participate. In many cases, these participants have no time, resources, or inclination to engage in research on the needs and practices

of their users. And, in many cases, these participants have no formal training in design, and hence no common perspective.

On the other hand, in a socio-technical environment the final frontier is not customization but it is creative participation. It means that user can intervene the flow of a program, but cannot change the face of an artifact. Creative participation (or user participation as it is called in socio-technical systems) is another driving factor for the diversification and complexity of this field.

Conclusion

The increasing complexity and diversity in the field of human-computer interaction can be a source of richness, or of chaos. One way to gain this richness is by creating a shared language for communicating insights and experiences to resolve common problems encountered throughout this field. By codifying these solutions and their relationships a body of knowledge will be captured which comprises an understanding of good architectures that meet the needs of both end users and system developers. The next chapter will introduce known approaches for creating a shared literature: **Standards and Patterns**.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 3:****Standard and/or Patterns****What is a Standard?****What is a Software Pattern?****Pattern Definitions**

<i>In the Dictionary</i>
<i>In Mathematics</i>
<i>In Architecture</i>
<i>In Software</i>

What is a Pattern Language?**What are Proto-Patterns?****Patterns as a Solution****Standards and/or Patterns**

*A Proto Pattern Language for Human-Computer Interaction***Chapter 3:****Standard and/or Patterns**

In the worlds of science and industry there are two known ways for creating a shared literature: 1) by using standards, 2) by using patterns. In this section I will look at the capabilities and /or weaknesses of each to solve the problem of the lack of a shared literature in the field of human-computer interaction. ¹⁹

What is a Standard?

For the high-tech industry, consistency across interfaces reduces learning time, improves performance, and reduces accidental errors. Consistency means that if the user learns how to use one service, she already knows how to use many of the related services. In order to bring consistency to many products and systems that are in development at any given time, the human factors departments of the high-tech companies have produced some evolving sets of design guidelines.

High-tech industries have attempted to base these guidelines on appropriate human factors research and experience, and on the best practices of the field. Those evolve as new research emerges, and as national and international standards bodies reach agreement on new standards. They also evolve as these

companies move into new areas of work. This means, of course, that these guidelines are not perfect.

For high-tech industries a ***standard is a guideline that is accepted by a standard body***. Therefore, ***today's guidelines may be the standards of tomorrow***. In the real world these guidelines are the property of the corporate. The scope of a guideline is limited to the corporate's products and the community of its employees.

A good example of the effort for creating a set of guidelines for human-computer interaction is the guidelines that are developed by Ameritech. The company has called those "standards and guidelines." Those are not standards since their scope is not national or international. This set includes four parts as follows:

1- Ameritech Graphical User Interface Standards and Design

Guidelines:

Robert M. Schumacher, Jr.

© Copyright Ameritech Corporation, 1992-1995

The purposes of the Ameritech graphical look and feel standards are twofold. First, it provides a basis for a consistent user interface look and feel across Ameritech applications and platforms. Second, it is the initial attempt to integrate research on user interface design directly into the

selection and usage of interface objects found in user interface libraries and toolkits. [Schumacher, 95]

2- Ameritech Character-based Interface Standards and Design

Guidelines:

Jayson M. Webb

© Copyright Ameritech Corporation, 1992-1995.

The goal of this document is to facilitate the evolution of a consistent look and feel for Ameritech character-based interfaces. A consistent look and feel benefits both end-users and developers. End-users benefit from being able to apply what they have learned from one product while using a different product. Developers benefit from a consistent approach to design that allows the re-use of design implementations, allowing them to spend more time solving novel problems. [Webb, 95]

3- Ameritech Phone-based User Interface Standards and Design

Guidelines:

Amy L. Schwartz, and Mary L. Hardzinski

© Copyright Ameritech Corporation, 1992-1995.

The goal of this document is to facilitate the evolution of a consistent style, or "look and feel" for Ameritech telephony products. Marketing studies have indicated that revenue is lost from existing services because their

services are not as easy-to-use as they could be. The development and adherence to a style guide for the user interface to their telephony products is one step toward ensuring that their new services will have a usable interface. The style guide is intended to, first, provide guidelines for service functionality that are based on established or well-researched human factors principles and practices. Second, a style guide for Ameritech telephony products will help ensure that these products have a consistent "look and feel." [Schwartz, et al., 95]

4- Ameritech Web Page User Interface Standards and Design

Guidelines:

Mark Detweiler Penn State University, ***Richard C. Omanson*** Ameritech
© Copyright Ameritech Corporation, 1996.

This document complements other Ameritech user interface guidelines, such as Ameritech's Graphical User Interface Standards and Design Guidelines. Its primary goal is to assist content providers in creating effective Web pages that have a consistent look and feel. Good interface design practices demonstrate working principles, examples, and checklists. These design practices offer techniques for making Web pages well organized, easy to navigate, and aesthetically pleasing. [Detweiler, et al., 96]

What is a Software Pattern?

Software patterns are a recent software engineering problem-solving field that emerged from the object-oriented community. Patterns have roots in many disciplines including mathematics -*The Science of Patterns* -, [Lynn, 88] and architecture.

Pattern Definitions:

In the Dictionary:

"Pattern" is defined as: "*an original or model considered for or deserving of imitation . . .*" [infoplease, 99-1]

In Mathematics:

"Pattern" denotes regularity in some dimension. The simplest examples are repeated visual units ordered with translational (linear) or rotational symmetry. Patterns also exist in a scaling dimension, where similar forms occur at different magnification. When geometric self-similarity is defined on a hierarchy of scales, a self-similar fractal is created. The concept of a pattern also extends to solution space, in that solutions to similar problems are themselves related and define a single template that repeats - with some variation - every time such a problem is solved. The underlying idea is to reuse information, whether in repeating a unit to generate a two-dimensional tiling design, or in reusing the general solution to a class of differential equations. [Salingaros, 99]

In Architecture:

Using patterns in art and architecture is an old tradition dating to mythological periods. Architectural patterns are the reflections of the application of the most ancient design theory: ***analogy***. Patterns in architecture were once evident to all people but which have become increasingly forgotten and neglected in the ***West*** since the advent of the Renaissance and among the modernized classes of the ***East*** since the spread of the modern mentality from the end of last century. [Nasr, 73]

During the 60's a new movement inspired by the idea of "returning back to the roots" as a reaction to what was considered as the "modern" artificial and harsh environment, brought back a notion of "pattern based architectural space" to the world of architecture parallel to post-modernism. Discussing the roots and diverse results and branches of this movement is out of the scope of this thesis. The only one, that this thesis will deal with, is Christopher Alexander's pattern theory, due to its impact on and popularity in the world of software.

Alexander's Pattern Theory is a rediscovery and systematization of the old analogous architectural patterns. In "***A Pattern Language***," [Alexander, et al., 77] Alexander presents problem solving processes from the scale of cities and cultures to construction and decoration. It is not a set of compatible "design modules" or a dictionary, but it is a toolkit. Indeed, it has enough flexibility and

possibility to justify its denotation as a language. Each pattern is a part of the vocabulary of the Alexandrian world, structured by a grammar relating patterns across scales.

Alexander's Pattern Theory has a dual concern. He extols the virtues of traditional artifacts: their "unselfconscious" design, their adaptability and beauty. Yet he proposes to achieve such integration through formal constraint analysis. In spite of this robotics picture, as Rohit Khare has evaluated, Alexander's pattern theory is conceptual kin to Louis Kahn's theories of "basic - concept - deformed - by - contextual - special - cases." [Khare, 95]

It is true that Alexander's Pattern Theory is a modernized (and more or less out of the context) version of the pattern based architectural space. This theory is important because it opens the door for discussion and exchange of ideas between software and architecture. Both of these disciplines focus on the spatial organization of our lives. Architecture defines our physical environment, while software builds the virtual one.

Christopher Alexander has defined patterns as the following: [Alexander, 79]

"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.

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 thing."

In Software:

Procedural and object-oriented approaches are two main streams of software programming. Object-oriented programming is particularly strong at designing applications that interact with the real world such as games, business processes, and productivity applications. Where procedural or "conventional" programming languages are like recipes, dictating a rigid series of instructions that mix and mash bits of data, Object-oriented programming uses objects to bind together particular data and the operations on them. So, it makes a clear analogy to a room full of people, each expert doing one thing with only the knowledge that person has. Things get done as people ask each other to compute certain things

for them. This encapsulation of behavior stands in contrast to the traditional "spaghetti " code of procedural programming. Object-oriented programming also holds up well in architectural metaphors. Objects can be assembled in complex structures just as in architecture, but with a time dimension that captures the ballet of messages and dynamic rearrangement of objects. [Khare, 95]

Object-oriented programming also holds up well by analogy to the real world. You can model a pencil as a pencil, a car as a car, and an invoice as an invoice. Furthermore, object-oriented programming lets us organize hierarchies of similar classes of objects. A go-cart is like a car, but smaller; a car, in turn, is like a Vehicle, but with four wheels and confined to the ground. We can introduce abstract concepts, like shape, to hold the behavior common to particular shapes, like circles, rectangles, and lines. [Khare, 95]

Design patterns are conscious attempts to bring Alexander's design theories over to the insubstantial world of software design. A core cult formed around Alexander in the early 90's and has blossomed into a movement in software design, with its own books, conferences, and constituency. Some of the main documents in software patterns literatures, however, deny impacts of Alexander's ideas. More likely this movement can be seen as Noam Chomsky's ²⁰ heritage, which uses some of the Alexandrian techniques.

In the realm of non-institutional architecture Chamsky is known because it is supposed that his ideas impacted on Eisenman's works. As Charles Jencks

noted in his book "The Architecture of the Jumping Universe," [Jencks, 97]

Eisenman's early work was based loosely on the theories of Noam Chomsky and his concept of deep structure. Eisenman's ²¹ work at this time was guided by his own reading of Chomsky's deep structures and generative grammars.

Eisenman tries to apply a theory with a target problem that is more complicated and broader than the target problems of his work. Therefore, his attempts have never received attention outside the realm of non-institutional architecture. Chomsky's theory has an organizational nature that can be applied at the level of institutional architecture where synthesis of forms reveals the organizational nature of architecture.

As I explained in Appendix 2 of my thesis, Alexander's pattern theory reveals the organizational ability of architecture by capturing the notion of synthetic architecture. Therefore, Alexander's ideas have received a proper recognition from those fields which are engaged with the organizational theories (software design, urban design and planning), and have become the key links for exchanging ideas between architecture and software.

Alexander's ideas found a comfortable home in software pattern movement, because many designers had already proposed convergent schemes, and because software design has always been in search of an established discipline to look up to. In the 1968, software engineering created by civil and electrical engineering, and inherited their statistical controls, standardized models, and absolute faith and correctness. Only recently has a new generation advocated adding more art to this pseudo science by advocating software architecture.

Software architecture as a professional metaphor works on several levels: as a client-services model, as an artistic pretension, as a central coordination point for the "construction" process, and a belated concession that there is some quality that separates elegant software from the rest. [Khare, 95]

In the Software Pattern Community, patterns are defined as follows:

Dirk Riehle and Heinz Zullighoven:

"A pattern is the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts." [Riehle, et al., 96]

Brad Appleton:

"A pattern is a named nugget of insight that conveys the essence of a proven solution to a recurring problem within a certain context midst competing concerns." [Appleton, 97]

Richard Gabriel:

"Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces, which occurs repeatedly in that context, and a certain software configuration, which allows these forces to resolve themselves." [Gabriel, 99]

James O. Coplien:

"I like to relate this definition to dress patterns. I could tell you how to make a dress by specifying the route of a scissors through a piece of cloth in terms of angles and lengths of cut. Or, I could give you a pattern. Reading the specification, you would have no idea what was being built or if you

had built the right thing when you were finished. The pattern foreshadows the product: it is the rule for making the thing, but it is also, in many respects, the thing itself." [Coplein, 96]

I think the most simple and straightforward definition for pattern is Richard Gabriel's definition. I will use this definition as a guideline.

Another concept that is closely related to design patterns and object-orientation is a software framework. A software framework is a reusable mini-architecture that provides the generic structure and behavior for a family of software abstractions, along with a context of memes / metaphors which specifies their collaboration and use within a given domain. [Appleton, 97] A brief review of this topic and its relation to patterns can be found in Part 2 of Appendix 5 of my thesis: "Related Patterns: Frameworks and Patterns."

As it is shown in Part 2 of Appendix 5 any application may have some frameworks and any framework may use some patterns or pattern languages. Therefore, the evaluation of the process of creating scenarios, use cases, capturing requirements and other related topics that are popular in Joint Application Design (or their equivalents in Participatory Design) is out of the scope of a pattern language and this thesis. ²²

What is a Pattern Language?

In the dictionary "Patterning" is defined as: "*A design or decoration formed*

by the creative arrangement or formation of patterns." [infoplease, 99-2] This definition is very similar to that which is used to define a pattern language in the software pattern community or by Alexander.

Christopher Alexander refers to a pattern language as follows:

"As in the case of natural languages, the pattern language is generative. It not only tells us the rules of arrangement, but shows us how to construct arrangements – as many as we want –, which satisfy the rules.

A pattern language gives each person who uses it, the power to create an infinite variety of new and unique buildings, just as his ordinary language gives him the power to create an infinite variety of sentences.

The structure of the language is created by the network of connections among individual patterns: and the language lives, or not, as a totality, to the degree these patterns form a whole.

Each pattern then, depends both on the smaller patterns it contains and on the larger patterns within which it is contained. The language is a good one; capable of making something whole, when it is morphologically and functionally complete. " [Alexander, 79]

Coplien defines a pattern language as follows:

"A pattern language defines a collection of patterns and the rules to combine them into an architectural style. Pattern languages describe software frameworks or families of related systems. [Coplien, 96]

A pattern language is a structured collection of patterns that build on each other to transform needs and constraints into an architecture." [Coplien, 99-2] Good pattern languages guide the designer toward useful architectures and away from architectures whose literary analogies are gibberish or unartful writing. Good architectures are durable, functional, and aesthetically pleasing, and a good combination of patterns can balance the forces on a system to strive towards these three goals. A good pattern language gives designers freedom to express themselves and to tailor the solution to the particular needs of the context where the patterns are applied. " [Coplien, 99-2]

This term is coined by Christopher Alexander and has never been used in the original versions of " pattern based architectural space." What that kind of architecture means by a system, style, school, or tradition is wider than this concept of the language. A pattern based architectural system covers not only a set of patterns in this field but the ability of those patterns to participate in an organized multi-layers system including technical to social and value sub-systems

that were woven together in the landscape of a culture.

What are Proto-Patterns?

Not every solution, algorithm, best practice, maxim, or heuristic constitutes a pattern. Even if something appears to have all the requisite pattern elements, it should not be considered a pattern until it has been verified to be a recurring phenomenon. Some feel it is inappropriate to decisively call something a pattern until it has undergone some degree of scrutiny or review by others.

If, as Alexander writes, a pattern must be both a process and a thing, then a pattern must describe not only the process that creates that thing, but also the thing created by that process. For this reason, many contend that patterns ultimately deal with some kind of visually discernible structure. You should be able to draw a picture of the kind of structure that results from using the pattern in practice. A "pattern in waiting" that is not yet known to pass the patternity tests mentioned above, or some of the ones mentioned below, is often called a proto-pattern. Proto-patterns are sometimes called patlets.

Standards and/or Patterns?

As I have mentioned, to create a set of standards there should be a guideline. Guidelines need to be approved by the official bodies to be accepted as standards and their application enforced. Patterns and guidelines belong to

the world of the corporate and bureaucracy. Brief descriptions of such patterns are shared vocabulary of a profession. Through patterns, professionals can teach their experiences to the newcomers to the profession. Also, through this shared literature they can talk to each other to find the best solution for a problem. This solution may be marketable. A market has its own principles and does not necessarily search for the best solution.

Patterns as a Solution

One way to create a shared language in the field of human-computer interaction is to build a pattern language. A language of this sort is a set of interrelated patterns, which share similar assumptions, terminology, and context.

Just like the natural languages that facilitate human communications inside cultural groups, a shared literature will increase the chance of participatory design, implementation and use, when all participants (software and interface designers, developers and users) can talk about the same concepts, in the same terms, with fewer misunderstandings. Also, a common shared language will create an opportunity to focus on essential values and unsolved problems. Such a mutual understanding and concentration on strategic problems would help the whole industry to develop better interfaces, tools and paradigms with less cost.

A pattern language will facilitate the process of capturing the collective wisdom of other experts in this field by experienced designers, less experienced

designers, developers and users. It will help **designers and developers** to build better interfaces by following the rules even without having a profound knowledge of the field. This language will create a proper atmosphere where **user participation** may really occur. It will also facilitate common dialogs between designers, multi-disciplinary design teams, developers and customers. Finally it will be a real help to decrease the **complexity** resulting from the multi-disciplinary nature of the field. If this language is organized to reflect its background software architecture as well as to facilitate user interventions, **transparency** will be achieved.

Transparency means that human-machine interface shouldn't be an interface between the user and the program inside the machine, but it should depict the structure of the program inside the machine. [Coplien, 98-1] This is important because users try to build their own mental models of what's going on inside the machine, based on their interactions with the interface. If the interface isn't a model of the program, then the designer has to figure out the mental model that the user crafts for the model of the program, and build the interface to that model. This interface, in turn changes the users' mental model, and the result will be an infinite loop of actions and reactions. This process will waste a lot of effort, time and money, and the final product may not be marketable.

An interface designer may consider working with the patterns as a constraint

especially those designers who come from artistic disciplines. But a pattern language can be flexible (generative) in regards to visual appearance and interaction details. To be generative means that the restrictions coded in the patterns of the language are as few as possible. Then a minimum functionality approach should be used to code those restrictions. This is the main difference between a pattern language and a template or framework where most possible restrictions are imposed. In response to general restrictions of a generative pattern language, new specific solutions may be achieved. Those solutions may be better than the others that are commonly found in a chaotic environment.

To find out a set of criteria and developing a strategy that can be used for defining a proto-pattern language for human-computer interaction, I will look at four main accessible sources regarding the patterns and pattern languages:

1. Original Pattern Based Architectural Space;
2. Christopher Alexander's Pattern Theory;
3. Software Patterns;
4. Human-Computer Interaction Patterns.

These sources are investigated through four sections as appendices of my thesis (Appendix 1 to Appendix 4). A brief review of the criteria that are extracted from these sources and are used in organizing my proto-pattern language will be included in the Part 1 of Section 1 of Chapter 4: " Background: Concepts."

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4:**

A Proto Pattern Language for Human-Computer Interaction

Section 1: Background

Section 2: A Proto Pattern Language for Human-Computer Interaction

Section 3: Patterns

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 1:**

Background

Part 1: Concepts

Part 2: Approach

Part 3: Choosing a Format

Part 4: Components of a Pattern

Part 5: Forces

Part 6: Sketch

Part 7: Conclusion

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 1: Part 1:

Concepts

I will borrow some basic ideas and concepts from:

**Pattern Based Architecture
Christopher Alexander's Pattern Theory
Software's Pattern Community
Human-Computer Interaction's Pattern Community**

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 1: Part 1:**

Concepts

This thesis is going to use the spatial organization and creative process of a "pattern based architectural space " ²³ as a spatial model for developing a pattern language for human - computer interaction.

I will borrow some basic ideas and concepts from the pattern based architecture, Christopher Alexander's Pattern Theory, and software patterns to implement a proto-pattern language for human-computer interaction, as well as from those that are specially related and developed in this field. The scope of this thesis does not cover the whole aspects of human-computer interaction. It is focused on the communicative aspects between human and computer.

In accordance with the goal of this thesis, I will consider any informative artifact, such as a book, a web page, or an educational software, as an informative spatial system. Then, I will borrow the following ideas from:

Pattern Based Architecture:

Any spatial system has an abstract spatial model (instead of a metaphor ²⁴), some spatial properties, and a reference point to locate its components and users

in the space. The spatial properties of the target system should have conformity with the corresponding properties of its abstract spatial model.

Any space has an organization. Space can be organized according to different orders based on its scale. Different orders result in different formations. Delimiters can produce a concrete representation of the abstract space through generic forms. Delimiters may be abstract, or may have a hybrid nature. Also, Generic Forms may be conceptual or physical.

The structure of a spatial system and its relation to the structure of the upper bound system (its universe) should be transparent to its user. This is "*the Right of the User*" to be informed of this structure and its position in the structural framework of its universe. The conformity of the system model, representation model and the user's mental view can be achieved by layering, not covering, the structure of the system. The expense of this conformity should not be uniformity. The goal is having maximum diversity as well as conformity.

The pattern based architectural space follows a low functionality approach by not imposing any restrictions or "*designing taste*" for using the space but those that are vital for its sustainability. Proper tools should be developed to facilitate user participation upon user request. Creative and unconditional participation of the user is one of the most important factors in the value system of a successful product.

I chose the pattern based architecture as the abstract spatial model of the

informative artifacts. I learned from this spatial model the process of synthesizing the forms to create a space. I defined the properties of the space of informative artifacts and reference point for that space in correspondence with those properties and a reference point in the pattern based architectural space. I defined the principles of creating different orders to organize the content of an informative artifact and navigate through that. I defined the delimiters and generic forms in the related context (an informative artifact) to break down the abstract organizational and conceptual patterns to visualized ones. Finally, I defined the principles of user participation in the informative space.

To summarize, *the proposed structure of the space of an informative artifact and the structure of my proto pattern language are inspired by the pattern based architecture as an abstract spatial model.*

Alexander's Pattern Theory:

The Alexander's triptych, "***The Timeless Way of Building***" (the manifesto), "***A Pattern Language***" (the technique), and "***The Oregon Experiment***" (the result) formed a forceful and complete argument for a new theory of design.²⁵ Indeed, Alexander's work to "rationalize" the artistic process of architecture is seen as often alienating to the practice of non-institutional architecture. Many of his ideas are accepted as norms in modern urban planning practice all over the

world. The user participation principle is an example. [UOPO, 92-1] The reaction of architectural community is the exact opposite of what later awaited the introduction of "artistic" patterns to the software engineering community. [Khare, 95]

As I have mentioned, Alexander's Pattern Theory is a modernized (and more or less out of the context) version of the pattern based architectural space. It is important because it opens the door of discussion and contribution between architecture and software. Both of these disciplines focus on the spatial organization of our lives.

To develop a modern version of my proto-pattern language, I borrow Alexander's approach in bringing a deprecated idea to the modern environment.

Alexander does not consider the "Oregon Experiment" as a successful example of the pattern based architecture. Indeed, Alexander simplified the complexity of the forces that are involved in the complicated and multi-layered system of a city in the pattern based architectural space, to those for a university campus. The result of this simplification was a template. There is not enough social and economic diversity and motivation in a campus that can stimulate the simultaneous processes of growth and development as there is in a city. In other words, in "Oregon Experiment" the order and context did not have conformity.

Software patterns have had a similar experience. The WindowPerTask pattern²⁸ is going to ask the user to determine the number of windows in an

application. The number of windows is actually subject to a complicated set of forces that should be resolved in different layers. What happens if in a large application a user divides her task to twelve thousand windows? The WindowPerTask pattern is not a general pattern, it works in special cases and this fact is not explained in its "context."

Managing complexity without simplifying it, is a good practical lesson to learn from Alexander's experience.

The pattern based architecture has a pattern language, but it does not call it "language." It is called "*ritual*." Alexander has coined the terms of Pattern and Pattern Language. Also, from an architectural viewpoint there is only one pattern language for each cultural entity. Alexander defines a pattern language as a set of patterns that can be synthesized.

The pattern's terminology will be borrowed from Alexander.

Software's Pattern Community:

There are many documented efforts to exchange the ideas between Architecture and Software especially through Alexander's idea.²⁷

I will use all vocabulary, terminology, and conventions from the software pattern community to write my proto-pattern language for human-computer Interaction.

Human-Computer Interaction's Pattern Community:

There are some pattern languages that have been written for this field. ²⁸

My proto-pattern language will benefit from them, and will borrow some of their patterns.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 1: Part 2:**

Approach

Architectural Approach
Christopher Alexander's Approach

What are Self-Organizing Systems?

Software Approach
Conclusion

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 1: Part 2:

Approach

Writing a Pattern language is not a "design" process, it is a "documentation" process. The documentation process also should follow a methodology.

To choose a methodology for developing my proto-pattern language I will review the popular architectural and software methodologies. Review of the contemporary approaches in the other domains is out of the scope of this thesis.

Architectural Approach:

The pattern based architecture examines three hierarchical steps in the externalization of the creative process: archetypes (meta - abstract), form (hybrid), and style (concrete). In this top-down hierarchical system forms are synthesized to create the space.

Christopher Alexander's Approach:

Alexander's pattern theory is a reference to "self organizing systems. Coplien refers to this fact when he wrote: "Alexander's process for applying patterns can be thought of as iterative or recursive, but the recursive perspective yields more insights into the geometric fruits of a pattern language. Each pattern works with finer structure than the one before it thus preserving the overall structure of the

whole. The result is fractal in nature. Alexander's later work, to be described in the forthcoming *Nature of Order*, more consciously aspires to this fractal nature."

[Coplien, 97-2]

Michael Beadle, author of *"Re-engineering the Application Development Process,"* likens the effects of using pattern languages to the generation of emergent behaviors: spontaneously recurring patterns of dense local interaction between entities, resulting in dynamic, self-organizing systems that are adaptive, open, and capable of multi-scale effects. [Beadle, 97]

What are Self-Organizing Systems?

Several mechanisms and preconditions are necessary for systems to self organize. They are more or less redundant and undefined, but they are useful intuitive indicators of the potential for self-organization. These mechanisms include:

Thermodynamically open: The system must be exchanging energy and/or mass with its environment.

Dynamic behavior: If a system is not at or near equilibrium then it is dynamic, meaning the system is undergoing continuous change of some sort. One of the most basic kinds of change for self organizing systems is to import usable energy from its environment and export entropy back to it.

Local interaction: All natural systems have inherently local interactions

and then self-organization must follow the same models.

Nonlinear dynamics: Self-organization can occur when feedback loops exist between component parts and between the parts and the structures that emerge at higher hierarchical levels.

Many parts: Self-organization needs the connections, interactions, and feedback loops between the parts of the system. Therefore, it is clear that a self organizing system must have a large number of parts.

Emergence: The theory of emergence says the whole is greater than the sum of the parts, and the whole exhibits patterns and structures that arise spontaneously from the parts.

Multi-scale effects: Emergence refers to the multi-scale interactions and effects in self-organized systems. The small-scale interactions produce large-scale structures which then modify the activity at the small scales.

Even knowing that self-organization can occur in systems with these qualities, it's evitable, and it's still not clear why it sometimes does. In other words, no one yet knows the necessary and sufficient conditions for self-organization.

To say that a system is self-organized is to say it is not governed by top-down rules, although there might be global constraints on each individual component.

Instead, the local actions and interactions of individuals are the source of the higher-level organization moving the system into patterned, ordered structures with recognizable dynamics. Since the origins of order in self organizing systems are the subtle differences between components and the interactions among them, system dynamics cannot be understood by decomposing the system into its constituent parts. Thus the study of self organizing systems is ***synthetic*** rather than ***analytic***. [Milne, 96]

Alexander's approach is an imitation of this theory. He applies a synthetic approach but he does not talk about omitting hierarchical relationships.

This is a lesson that he has learned from pattern-based architecture.

Software Approach:

In computer terminology bottom-up (data driven) processing stresses the importance of information from the stimulus (structured programming). On the other hand, in Top-down (conceptually driven) processing knowledge of the world helps in identifying patterns (object oriented programming).

The software developers often use one of the following models for developing applications: ***Waterfall model, Spiral model, or Fractal model.***

The ***Waterfall model*** of the software life cycle as shown in Figure 4, incorporates some re-engineering (dashed lines). There is a constant flow from one block to the next as well as a continuous assessment to see how well the previous sub-task has been performed and, if necessary, a return back to modify that module.

Figure 4 : Waterfall model of application development in software design

The **Spiral model** or object-oriented model for software engineering (Figure 5) has evolved to encompass the best features of the classic waterfall model, while at the same time adding an element known as risk analysis. The spiral model is more appropriate for large, industrial software projects and has four main blocks/quadrants.

Each release or version of the software requires new planning, risk analysis, engineering and customer evaluation phases as illustrated in the model by the spiral evolution outwards from the center. For each new release of a software product a risk analysis audit should be performed to decide whether the new

objectives can be completed within budget (time and cost) and decisions have to be made about whether to proceed. The level of planning and customer evaluation is missing from the waterfall model that is mainly concerned with small software programs. The spiral model also illustrates the evolutionary development of software where a solution may be initially proposed which is very basic (first times round the loop) and then later releases a new feature added and possibly more elaboration done. [Brown, 97]

Figure 5: *Spiral model of application development in software design*

The **Fractal Model** (Figure 6) distinguishes three distinct stages, or phases, in the evolution of object-oriented abstract classes, frameworks, and components. [Foote, 91]

1- Prototype or Initial Design Phase,

2- Expansionary or Exploratory Design Phase,

3- Consolidation or Design Generalization Phase.

Figure 6 : *Fractal model of application development in software design*
A node within a node within a node . . .

The Fractal Model is an opportunity driven model, as opposed to the Spiral Model, which is risk driven. At each level, the model moves from the prototype phase through the expansionary phase into the consolidation phase. From there, an element may expand further or serve as the basis for a new prototype. As objects evolve, the breadth of class hierarchies decreases as the depth increases, and discrete, black box components emerge from ill-defined inheritance hierarchies. Both progressions enhance the system's generality and reusability. The same pattern is repeated at every level: within individual classes and applications, up into the elements of the framework that can encompass a

family of related applications.

Conclusion:

Coplien has written: "A pattern language defines a collection of patterns and the rules to combine them into an architectural style. Pattern languages describe software frameworks or families of related systems." [Coplien, 99-1] But what is the methodology for developing those? There are some descriptions for what should be included in it, how it will improve during the time, and so on. [Meszaros, et al., 97]

In my thesis I will follow the approach of pattern-based architecture to organize my proposed proto-pattern language. This implies that I will use a top-down synthetic approach.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 1: Part 3:

Format

Different Formats
Choosing a Format

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 1: Part 3:

Format

Different Formats:

There are different formats for writing a pattern. The most popular ones include the following: ²⁹

- 1. Alexandrian Format,** (as shown in Figure 7)
- 2. Portland Format,** (as shown in Figure 8)
- 3. Modern Alexandrian Format,** (as shown in Figure 9)
- 4. Canonical Format,** (as shown in Figure 10)
- 5. Gang of Four Format,** (as shown in Figure 11)
- 6. Pattern-Oriented Software Architecture Format,** (as shown in Figure 12)

Gang of Four and Pattern-Oriented Software Architecture Formats are purely focused on the software design issues. Alexandrian and Portland Formats are very general. The Canonical and Modern Alexandrian Formats are more or less similar and can cover different disciplines. They are more general than Gang of

Figure 7: Alexandrian Format

If :	you find yourself in CONTEXT
For example	EXAMPLES
with	PROBLEM
entailing	FORCES
THEN: for some	REASONS
apply	DESIGN FORM and/or RULE
to construct	SOLUTION
leading to	NEW CONTEXT and other Patterns

Figure 8: Portland Format

Context:

Having done so and so you now face this Problem.

Rationale:

Here is why the problem exists and what Forces must be resolved.

Therefore:

Make something along the following lines. I will give you the help I can (Solution)

Now you are ready to move onto one of the following problems (Resulting Context)

Figure 9: Canonical Format

Name _____

Alliances

Problem

Forces

Context

Solution

Resulting Context

Rationale

Known Uses

Related Patterns

Sketch

Authors

Date _____

Email

References

Keywords

Examples

Figure 10: New Alexandrian Format

Name

Problem

Forces

Context

Solution

Examples

Resulting Context

Rationale

Related Patterns

Known Uses

Four and Pattern-Oriented Software Architecture Formats and more clear and organized than Alexandrian and Portland.

The mandatory parts of a pattern are only these four sections: Problem, Context, Forces, and Solution. [Meszaros et al, 97]

Figure 11: Gang of Four Format	Figure 12: Pattern-Oriented Software Architecture Format	Figure 13: The format that will be used for writing the patterns of my proposed proto-pattern language
Pattern Name	Pattern Name	Name
Scope, Purpose		(Also known as:)
Intent	Also Known As	Problem
Also Known As	Example	Context
Motivation	Context	Forces
Applicability	Problem	Solution
Structure	Solution	Rationale
Participants	Structure	Resulting Context
Collaborations	Dynamics	Known uses
Consequences	Implementation	Related patterns
Implementation	Example	Sketch
Sample code and usage	Resolved	Examples
Known Uses	Variants	
Related Patterns	Known Uses	
	Consequences	
	See Also	

Choosing a Format:

I will use Canonical Format to develop the patterns of my proto-pattern language with some changes. The resulting format is shown in Figure 13. The reasons for this choice are:

1. A goal of developing this proto-pattern language is to experiment if the principles of pattern-based architecture are applicable in developing an

informative artifact. Therefore, using Canonical or Modern Alexandrian Formats that are more or less similar seems more suitable in the case of my proposed proto-pattern language. Since these formats are familiar for both involved disciplines (Architecture and Software), using one of those formats in this case can help to communicate the ideas.

2. Canonical Format follows a logical principle that will omit the repetition of descriptions and reasoning throughout the body of pattern and will make it easier to follow the content of the pattern.

3. The first pass of writing and reviewing the patterns showed me that putting the "Rationale" heading of a pattern just after the solution heading reading works better.

4. A sketch as Coplien has described [Coplien, 98-1] is added as an optional section to give a clearer image of the concept. It will be considered optional since sometimes it is impossible to draw a meaningful picture of an abstract concept.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 1: Part 4:**

Components of a Pattern

Name
Problem
Context
Forces
Solution
Rationale
Resulting Context
Known Uses
Related Patterns
Sketch
Examples

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 1: Part 4:**

Components of a Pattern

As I have mentioned in part three of this section, for writing the patterns of my proto-pattern language I will use a Canonical Format and will add a heading (sketch) from the New Alexandrian Format to its headings. Therefore, the essential headings of the patterns in my Proto-Pattern Language include: ³⁰

Name:

Any pattern should have a meaningful name. That is a single word or short phrase to refer to the pattern and the knowledge and structure it describes. It would be very unwieldy to have to describe or even summarize the pattern every time when it is used in a discussion. Good pattern names form a vocabulary for discussing conceptual abstractions. Sometimes a pattern may have more than one commonly used or a recognizable name in the literature. In this case it is a common practice to document these names or synonyms under the heading of Aliases or Also Known As. Some pattern forms also provide a classification of the pattern in addition to its name.

Problem:

A statement of the problem which describes its intent, the goals and objectives it wants to reach within the given context and forces. Often the forces oppose these objectives as well as each other.

Context:

The preconditions under which the problem and its solution seem to recur and for which the solution is desirable should be described under this title.

This tells us the pattern's applicability. It can be thought of as the initial configuration of the system before the pattern is applied to it.

Forces:

A description of the relevant forces and constraints and how they interact or conflict with one another and with goals we wish to achieve. There may be some indication of their priorities. A concrete scenario which serves as the motivation for the pattern is frequently employed. Forces reveal the intricacies of a problem and define the kinds of trade-offs that must be considered in the presence of the tension or dissonance they create. A good pattern description should fully encapsulate all the forces which have an impact upon it. ³¹

Solution:

Static relationships and dynamic rules describing how to realize the

desired outcome will be included. This is often equivalent to giving instructions which describe how to construct the necessary work products. The description may encompass pictures, diagrams and prose which identify the pattern's structure, its participants, and their collaborations to show how the problem is solved. The solution should describe not only static structure but also dynamic behavior. The static structure tells us the form and organization of the pattern, but often it is the behavioral dynamics that make the pattern "come alive." The description of the pattern's solution may indicate guidelines to keep in mind when attempting a concrete implementation of the solution. Also, solution should describe the pitfalls to avoid. Sometimes possible variants or specializations of the solution are also described.

Rationale:

A justifying explanation of steps or rules in the pattern and also of the pattern as a whole in terms of how and why it resolves its forces in a particular way to be in alignment with desired goals, principles, and philosophies. It explains how the forces and constraints are orchestrated in concert to achieve a resonant harmony. This tells us how the pattern actually works, why it works, and why it is "good." The solution component of a pattern may describe the outwardly visible structure and behavior of

the pattern, but the rationale is what provides insight into the deep structures and key mechanisms that are going on beneath the surface of the system.

Resulting Context:

The state or configuration of the system after the pattern has been applied, including the consequences of applying the pattern, and other problems and patterns that may arise from the new context. It describes the post conditions and side effects of the pattern. This is sometimes called resolution of forces because it describes which forces have been resolved, which ones remain unresolved, and which patterns may now be applicable. Documenting the resulting context produced by one pattern helps you correlate it with the initial context of other patterns. A single pattern is often just one step toward accomplishing some larger task or project.

Known Uses:

Describe known occurrences of the pattern and its application within existing systems. This helps validate a pattern by verifying that it is indeed a proven solution to a recurring problem. Known uses of the pattern can often serve as instructional examples.

Related Patterns:

The static and dynamic relationships between this pattern and others within the same pattern language or system will be included. Related patterns often share common forces. They also frequently have an initial or resulting context that is compatible with the resulting or initial context of another pattern. Such patterns might be predecessor patterns whose application leads to this pattern; successor patterns whose application follows from this pattern; alternative patterns that describe a different solution to the same problem but under different forces and constraints; and co-dependent patterns that may or must be applied simultaneously with this pattern.

Sketch:

The sketch (a drawing) shows not only the major relationships between parts, which is the facet of patterns that perhaps most interested the object-oriented software community, but it relates directly to human sense of aesthetics. People share the cognitive processes by which they appreciate beauty and the sketch appeals to that universal sense of aesthetics. The software designer uses design diagrams based on standard notations. Alexander's sketches do not use OMT, UML, or Booch notations. OMT diagrams don't easily compose, whereas sketches do. Such composition is

fundamental to piecemeal growth and the current notations of software design are antithetical to such an approach to software. ³²

Examples:

One or more sample applications of the pattern which illustrate: a specific initial context; how the pattern is applied to and transforms that context; and the resulting context. Examples help the reader understand the pattern's use and applicability. Visual examples and analogies can often be especially illuminating. An example may be supplemented by a sample implementation to show one way the solution might be realized. Easy-to-comprehend examples from known systems are usually preferred.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 1: Part 5:**

Forces

System Collaborations

User Collaborations

Conclusion

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 1: Part 5:**

Forces

System Collaborations:

The notion of force in software design generalizes the kinds of criteria that software engineers use to justify designs and implementations. In the classic study of algorithms in computer science the main force to be resolved is efficiency (time complexity). However, patterns deal with the larger, harder-to-measure, and conflicting sets of goals and constraints encountered in the development of every artifact. Software design classifies these criteria in five groups including: correctness, resources, structure, construction, and usability considerations. Actually, all of these considerations cannot be regarded in a product or solution. Software designers try to find a solution to optimize the negative impacts of one of them on the other. A good solution in software design should balance these different considerations. The main considerations that are included in each of these groups are as follows: ³³

Correctness Considerations: Under this category software supports the completeness and correctness, static and dynamic type safety, multithreaded safety and liveness, fault tolerance and transactionality,

security and robustness of the solution and the product.

Resource Managements: Under this category software supports efficiency (such as performance, time complexity, number of messages sent, bandwidth requirements), space utilization (such as number of memory cells, objects, threads, processes, communication channels, processors), incrementalness (on-demand-ness), policy dynamics (such as Fairness, equilibrium, stability) of the solution and the product.

Structural Considerations: Structural considerations in software design support modularity (encapsulation, coupling, independence), extensibility (subclassability, tunability, evolvability, maintainability), reusability (openness, composability, portability, embeddability), context dependency, interoperability, and all "quality factors" of the solution and the product.

Constructional Considerations: Constructional considerations in software design support understandability (minimality, simplicity, elegance) error-proneness of implementation, coexistence with other software, maintainability, impacts on/of development process, impact on/of development team structure and dynamics, impact on/of productivity, scheduling, cost Impact on/of user participation for the solution and the product.

Usability Considerations: Under this category software supports ethics

of use, human factors (learnability, undoability) adaptability to a changing world, aesthetics, medical and environmental impact, social, economic and political impacts and other impacts on human existence.

Any solution may have some conflicts with some of these considerations or provide a better support for some of them or have no impacts on the others. In this view any of these considerations that are affected by the solution should be treated as a force. A better solution (pattern) is that solution which can resolve most of these conflicts and activate most of those supportive features.

User Collaborations:

Most of the time products are tested based on "**Common Sense**" criteria. As long as we use a computer program as a technical tool this approach may work. In the background of this approach, there is a notion of "implementation hiding," where a user is considered as a consumer not a participant. All the criteria that are mentioned above reflect this point of view.

This approach is not sufficient when the program is going to talk to the public. One of the main points of the application of a pattern language for Alexander is to open the door for **user participation**. Alexander extracted this idea from the old paradigm of the pattern based architectural system.

Computer technology is often out of reach for ordinary users. It seems that most applications are designed to be used by its developers. Changing this

attitude is not an arbitrary choice but from an economic viewpoint is a need for industry. Computer science involves a psychological view to evaluate human factors where **generality** is the most important factor. The user participation paradigm is looking for a social view where **diversity** and **generality** work together. User participation does not mean only usability of the artifact but goes further to examine its accessibility and usefulness assuming that usability is a basic property of the artifact. A new way of thinking about computer technology is on the way: "Think of Computers as a Medium, not as a tool." [Laurel, 93]

Brenda Laurel proposes an interesting view of interactive applications. She compares drama to user interfaces in that the user performs dramatic events in interacting with the application software and hardware. A computer user acts out a role in a play that carries out a predefined task or series of actions which are easily understood in simulations and virtual reality applications design. This is especially true if we consider the navigator/reader as one character among others in our hypermedia document.

Brenda Laurel views a theatrical production as a metaphor for a computer program. Coplien emphasis's is that a computer program should be emotionally engaging and emotionally satisfying. Using a program to accomplish some task should leave the user feeling somehow fulfilled. In the process of doing a task, a user wants to feel engagement with the computer and more specifically with the

computer program. That means that the human/machine interface shouldn't be an interface between a user and the program inside the machine but that it should depict the structure of the program inside the machine. Users try to build models of what's going on inside the machine based on their interactions of the interface. If the interface isn't a model of the program then the designer has to figure out the model the user craft for the model of the program and build the interface to that model. That in turn changes the users model.

Agents and objects can lead, tell, present, represent, act or create a navigational path. A user can view documents as having some sense of their own. And very often there may be some sort of agent involved who checks out how the document is browsing and how it is approached. If there are parts that seem too difficult and produce too many errors the agent may start alleviating this problem by rearranging the story line. [Salo, 94]

The following are some examples of conflicting sets of goals and constraints encountered in applying a user participation policy in an informative artifact: ³⁴

Intervention and Interactivity Considerations: An informative artifact should facilitate the creative and formal participation of its user in creating and changing its structure and space. To do so, the artifact should be interactive, customizable, and follow a minimum functionality approach versus a maximum functionality approach. Also, the artifact should support

the diversity of its look and feel and suggest an intelligent response to the user. The artifact should correct the minor user's mistakes and be helpful on her major mistakes.

Transparency Considerations: An informative artifact should present a clear view of the system functionality, and the user's location in the artifact (sense of place.)

To be Upon Request:

An informative artifact should present the minimum amount of information, tools, and ability of the system as default. It should represent other options and its abilities only if it is requested.

Usability issues:

In an informative artifact the effort for learning and the duration of use of that knowledge should be balanced. It should have an understandable structure, harmony and continuity.

These issues are not included in any categories of software design considerations for developing a good solution or product. As I explained in the previous section, any of these considerations that are affected by the solution for developing an informative artifact should be treated as a force.

Conclusion:

Based on the results of two previous sections, my proto-pattern language will

deal with the forces that are summarized in Table 2. Table 2 shows the conflicting sets of goals and constraints encountered in the development of every informative artifact that is going to follow a user participation policy.

Domain	Considerations (Forces)
System Collaborations	Correctness, Resources, Structural, Constructional, and Usability Considerations
User Collaborations	Intervention and Interactivity, Transparency Considerations, To be Upon Request

Table 2: *Sets of goals and constraints encountered in the development of an informative artifact that follows a user participation policy.*

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 1: Part 6:

Sketch

What is an Sketch?
Sketch in Software Design

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 1: Part 6:

Sketch

If you can't draw a picture of it, it isn't a pattern. [Alexander, 79]

What is a Sketch?

Architects are familiar with the importance of a sketch to explain the rationale of a design and how it can help to shorten an otherwise long discussion. By drawing a sketch the next step in the design process seems to come more into focus. A sketch is not a blue print and is a traditional tool for architecture. Patterns have never used blue prints but have a long history of using sketches.

Alexander and Coplien emphasize the importance of drawing a sketch for a pattern.³⁶ The sketch shows not only the major relationships between parts, which is the facet of patterns that perhaps most interested the object-oriented software community, but it relates directly to human sense of aesthetics (and *curiosity*).³⁸ People share the cognitive processes by which they appreciate beauty, and the sketch appeals to that universal sense of aesthetics (and *curiosity*).

The software designer uses design diagrams based on standard notations.

Alexander's sketches do not use OMT, UML, or Booch notations. OMT diagrams don't easily compose (and *extract from each other*), whereas sketches do. Such composition (and *extraction*) is fundamental to piecemeal growth and the current notations of software design are antithetical to such an approach to software.

Alexander's early work [Alexander, 84] portrayed system structure in a simple set of composable diagrams. Each element not only captured a geometric structure but encoded requirements addressed by the element. For example, in his work on Indian villages, he drew some sketches to represent a village as follows:

Figure 14 demonstrates a farm that embodies requirements such as getting the best cotton and cash crop, the best food grain crop and a good vegetable crop. It reflects the fact that crops must be brought home from the fields and encapsulates a lot of information about the farm.

Figure 14: A Farm

Figure 15 represents a water distribution system for the fields. It captures

requirements such as the need to divide land among sons of successive generations, the desire of people to own land personally, cooperative farming, maintenance of irrigation facilities and abolition of Zamindari and uneven land distribution in an Indian village.

Figure 15: A water distribution system

Figure 16 is a water collection unit built in the highest corner of the village at right angles to the terrain. It's designed to reclaim uncultivated land, to develop horticulture, to accommodate full collection of underground water for irrigation and monsoon water for use, to drain the land to prevent water logging, to support healthy plant ecology, and to address road and dwelling erosion. The small filled circles are springs or wells.

Figure 17 shows that how the previous figures can be combined or synthesized to represent a village.

Figure 16: A water collection unit

Figure 17: A Village

This figure encapsulates all of the information that is embedded in the previous sketches and also the information about their combination as a whole. Each drawing isn't just a sketch but has behind it a set of requirements (*Information*) that come together in the structure representing it. The structure of a village brings together these elements to minimize conflict between requirements (*Conflict resolution in the procedure of the reveal of information*). Alexander guides this integration by making a graph of the dependencies between requirements (*Information*). The village structure emerges from the graph. That is, there is a process for assembling these

ways, a process that is germane to the elements themselves (*The procedure of reveal of information*).

This is analogous to principles deep beneath the object paradigm. Parnas' original work on modularity emphasized that modules hid "design secrets." [Parnas, 78] Parnas' work is similar to the way Alexander's pictures embody design constraints.

Sketch in Software Design:

Coplien shows how drawing a sketch can help to organize a pattern and how this principle can be applied in software. A series of diagrams are drawn to represent the concept of Gerard Meszaros' Half-Object + Protocol (HOPP) pattern. [Meszaros, 95] This pattern is one of the best examples of a good pattern, not only in its form and presentation but also as an important design technique. Figures 18, 19 and 20 show the evolution of these diagrams. [Coplien, 98-1]

Figure18: Half-Object + Protocol

Figure 19: *Half objects are asymmetric*

Figure 20: *Adding more Half Calls to the Call*

The common understanding in both fields (Architecture and Software) indicates the value of a sketch in developing an idea one step further. Thus the "Sketch" heading is added to headings of patterns of my proto-pattern language.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 1: Part 7:

Conclusion

Chapter 4: Section 1: Part 7:

Conclusion

My Proto-Pattern Language will be developed based on the following conclusions that have been drawn from the previous sections of this chapter:

1. Any informative artifact is an **informative spatial system**.
2. In my prot-pattern language the abstract spatial model of the informative artifact is the **pattern based architectural space**.
3. My proto-pattern language defines the spatial properties and a reference point to locate the components and the users in the space of the informative artifact in accordance with the similar properties of the model. To do this task, my proto-pattern language will start with a group of patterns that covers all spatial considerations of the informative artifact.
4. My proto-pattern language uses the same methodology that the model applies to realize its abstract spatial values. Different orders for organizing the space of an informative artifact, based on its scale and the resulting formations should be evaluated. To do this task my proto-pattern language should include a group of patterns that covers all

organizational considerations of the informative artifact.

5. By following the same methodology that the model uses for realization of the abstract spatial values, my proto-pattern language should include a group of patterns that delimit its abstract values and produce a concrete representation of its abstract space through generic forms.
6. My proto-pattern language includes a group of patterns that delimiters can use to create a concrete representation of the abstract values.
7. My proto-pattern language includes a group of patterns that facilitate the participation of the user in creating and changing the space of the artifact. This language should pay special attention to transparency and low functionality principles as the most important features that can facilitate user participation.
8. I will use Christopher Alexander's approach to develop a modern version of my proto-pattern language. I also will use his pattern's terminology and will remember his practical lesson about "managing complexity without simplifying it."
9. My proto-pattern language uses all the vocabulary, terminology, and conventions of the software pattern community.
10. I use some of the patterns from pattern languages that have been written for human-computer interaction.

- 11. To organize my proto-pattern language I use a top-down synthetic approach.**
- 12. To write my proto-pattern language I use a version of Canonical Format as shown in Figure 13 (page 77).**
- 13. To write my proto-pattern language, I focus on balancing the impacts of the solution on the sets of goals and constraints that are shown in Table 2 (page 93).**

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 2:**

A Proto Pattern Language for Human-Computer Interaction

Introduction
Proto-Pattern Language Summary
How to Use these Patterns
Conventions
Problem/Solution Summary

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 2:**

A Proto Pattern Language for Human-Computer Interaction

Introduction:

My proposed proto-pattern language uses the pattern based architectural space as a model for creating the space of an informative artifact (see conventions heading of this section). It will follow the same procedure for extracting a representation from an abstract model as the pattern based architecture does when it tries to synthesize the forms. I learned from Christopher Alexander how to modernize the concept of "pattern" and borrow from him the terminology of a pattern language and modern principle of user participation. I use shared vocabulary of the software pattern community and benefit from the methodologies that they have used to transfer the experiences and ideas of other fields of human knowledge to the world of software design. I use some of the patterns of those pattern languages which have as their primary concerns human-computer interaction issues.

This proto-pattern language does not cover all aspects of human-computer

interaction. It is focused on the "communicative aspects" of the relationship between the user and computer. ³⁷

Proto-Pattern Language Summary:

The patterns in this proto-pattern language are grouped as follows:

- 1. Spatial Patterns,**
- 2. Organizational Patterns,**
- 3. Delimiter Patterns,**
- 4. Generic Forms Patterns,**
- 5. User Participation Pattern.**

Group 1: Spatial Patterns: introduce the concepts of *model*, *space*, and *time* for an informative artifact. They have an abstract nature.

Group 2: Organizational Patterns: introduce the concept of *organization* for an informative artifact. They have an abstract nature.

Group 3: Delimiter Patterns: introduce the concept of *realization* for an informative artifact. They have a hybrid (abstract / concrete) nature.

Group 4: Generic Forms Patterns: introduce the concepts of technical and pictorial *widgets* for an informative artifact. These patterns have a concrete nature.

Group 5: Social Patterns: introduce the concept of *user participation* for an informative artifact. This pattern has a social nature.

Each section starts with a brief summary, which introduces the patterns described in the section. The patterns in the entire collection are depicted graphically in Figure 21 and summarized later in the Tables 3 to 7 of Problem / Solution Summary heading of this section.

Figure 21: Structure of the proposed proto-pattern language

How to Use These Patterns:

If you are going to use this proto-pattern language and you are searching for a solution to a particular problem in the field of human-computer interaction, please follow these steps:

- 1- Refer to the Problem/Solution Summary tables to see if any of the

problems resemble the one you are trying to solve.

2- Since each pattern has been written with skip-able headings you could flip through the language looking at the Name, Problem, and Solution headings.

3- Once you have determined that a pattern is of interest you can look at the Context and Forces headings for guidance on determining whether it is applicable to the situation.

4- For more information you can look at the Rationale, Resulting Context, Related Patterns, and Examples headings to get further appreciation of the nuances of the pattern.

5- Finally if you are interested to look at the procedure of developing the language you can refer to the whole content of this thesis.

Conventions:

Any new or obscure term in the pattern language or its patterns is defined at the glossary heading of my thesis.

All references and notes of pattern language or each of its patterns are included in the notes heading of my thesis.

Also, all references of pattern language or each of its patterns are included in the notes and references headings of my thesis.

Throughout my thesis and this pattern language when I use "Architecture" or any related terms (such as architectural space) I mean institutional architecture or a reference to that. Pattern based architecture is not the only form of institutional architecture. Therefore, I indicted this category where I referred to.

Problem/Solution Summary:

The following tables (3 to 7) will summarize the patterns in this proto-pattern language for reference purposes.

Table 3: Spatial Patterns

Problem	Solution	Name	No
What is the spatial model of an informative artifact?	Use an abstract interpretation of architectural space as a model for the space of your informative artifact. Use the <i>Spatial Pattern</i> to define the correspondents of the spatial properties of the model for the space of your artifact.	Spatial model Pattern	01
How should the properties of the abstract spatial model be applied to the space of the informative artifact?	Develop a spatial strategy for creating the space of your informative artifact by defining correspondents for eight main spatial properties of the model. Use <i>Organizational Patterns</i> to implement this strategy.	Spatial Pattern	02

Table 4: Organizational Patterns (part 1)

Problem	Solution	Name	No
What are the possible configurations for organizing the content of an informative artifact?	There are three main possible configurations for organizing the content of an informative artifact that may be combined. Any mixed configuration should have a predominant order. Use the <i>Scalability Pattern</i> to see which configuration should be applied	Formation Pattern	03
What is the impact of size on choosing an order for an informative artifact?	Use simplest order and consider the graded system last. Reorganize the content to reduce the complexity.	Scalability Pattern	04

Table 4: Organizational Patterns (part 2)

Problem	Solution	Name	No
How can you create a graded informative artifact?	Create a sequence of sequences. Use the <i>Sequence Pattern</i> to create any of these sequences.	Grading Pattern	05
What are the basic configurations of navigational paths in an informative artifact?	Create a set of distinguished paths including a primary and a secondary subset. Create a dynamic main path of navigation as a part of the primary set of the navigational paths.	Navigation Pattern	06
How do you put together the necessary information that can explain an independent part of the content of an informative artifact?	Create a sequence. Manage changes, recovery process, state transitions, and a return path for that sequence. Insert the sequence in the artifact.	Sequence Pattern	07
How should a user return to the upper level sequences or exit from a graded informative artifact?	Create a short path independent of the navigational path, by connecting all main nodes of all sequences of the artifact.	Short-Cut Pattern	08
How do you manage the transition between and through different levels of a sequential graded informative artifact or the artifact and the background system?	Create a transitional subsystem for your artifact. Manage active grade transitions, passive grade transitions, shortcut transitions, and the impact of changes on the navigational path. Insert the transitional subsystem in the artifact.	Transition Pattern	09
What are the default values of an informative artifact?	Apply a low functionality approach for creating default values. Create a small set of elaborate default values with minimum diversity and a small set of elaborated configurations with maximum diversity for each default value and consider those as preferences.	Default values of an informative artifact.	10
What is the state of a main node when a user enters to it before or after navigating a sequence of a sequential informative artifact?	Classify states in very general categories. Develop a multiple representation of a main node. Elaborate your artifact by creating an organizational indicator.	State Pattern	11
How long should your informative artifact wait for a user interaction before doing the job itself?	Delay in a short-cut path is shorter than delay in a navigational path. Use an organizational indicator in both cases. Test the length of delay with many users.	Delay Pattern	12
Can you apply the organizational patterns to an informative artifact with a non-sequential formation?	You can apply the <i>Formation Pattern</i> , the <i>Scalability Pattern</i> , the <i>Delay Pattern</i> , and (in some cases) the <i>State Pattern</i> .	non-Sequential Pattern	13

Table 5: Delimiter Patterns

Problem	Solution	Name	N°
How much of the content of an informative artifact should reflect in a window?	Use the <i>Task Window Pattern</i> . Use correspondent patterns of this proto-pattern language instead of related patterns of the <i>Task Window Pattern</i> .	Window (Shape) Pattern	14
How do you represent the content of a window of an informative artifact, from several points of view?	Use the <i>Dividing the Window into a Few Panes Pattern</i> . Use pictorial patterns of the visual arts to compose your panels.	Panel (Surface) Pattern	15

Table 6: Generic Forms Patterns

Problem	Solution	Name	No
How do you create a visual representation of your informative artifact using conceptual widgets?	Use " <i>A Pattern Language for Human-Computer Interface Design</i> " and " <i>Iconic Reference Pattern</i> ."	Conceptual Widgets Pattern	16
How do you create a visual representation of your informative artifact and conceptual widgets?	Read the specification of the technical (programming) language of your artifact, to have a general idea of its abilities and restrictions.	Concrete Widgets Pattern	17

Table 7: Social Patterns

Problem	Solution	Name	No
How do you facilitate the participation and intervention of the user in an informative artifact?	Work with the user. Create a variety of solutions. Use a minimum functionality approach. Implement your findings through the process of creating your artifact.	User Participation Pattern	18

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3:**

Patterns

Part 1: Spatial Patterns

Part 2: Organizational Patterns

Part 3: Delimiter Patterns

Part 4: Generic Forms Patterns

Part 5: Social Patterns

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 1:

Spatial Patterns

This set of patterns refers to the spatial modeling and space of an informative artifact.

N	Problem	Solution	Name	N*
01	What is the spatial model of an informative artifact?	Use an abstract interpretation of architectural space as a model for the space of your informative artifact. Use the <i>Spatial Pattern</i> to define the correspondents of the spatial properties of the model for the space of your artifact.	Spatial model Pattern	01
02	How should the properties of the abstract spatial model be applied to the space of the informative artifact?	Develop a spatial strategy for creating the space of your informative artifact by defining correspondents for eight main spatial properties of the model. Use <i>Organizational Patterns</i> to implement this strategy.	Spatial Pattern	02

* The number of a pattern in my proto-pattern language.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 1: Pattern1:**

Spatial Model Pattern

Name

Spatial Model Pattern

Metaphor

Problem

What is the spatial model of an informative artifact?

Context

You want to design a navigable and interactive informative artifact. You may use an informative model to implement the space of your artifact. The other models that you can use include a theatrical model [Laurel, 93], a cinematographic model, a pictorial model, a verbal model (like a book), or an architectural model.

Forces

Any solution to this problem should balance the following forces:

Capability of nominated models: There is no spatial informative model that you can use to implement the space of your informative artifact.

Therefore, you should refer to the experiences of the other nominated models. The informative space should be navigable and interactive. The

cinematographic, theatrical, and pictorial spaces are not navigable and interactive. The verbal model is navigable, but is not interactive. The architectural model is navigable and interactive, and uses an organizational approach to manage its changeable space.

Conflict with the model: Architecture has a real and physical space. You live in an architectural space physically and mentally. But you can live in an informative space only mentally. Also, the architectural space and the informative space have different lifetimes. The lifetime of architecture is historical, but information can change on the fly.

Solution

Use an abstract interpretation of the architectural space as a model for the space of your informative artifact. In architecture the space has eight main properties.³⁸ Use the *Spatial Pattern* to abstract, interpret and implement the parallel spatial properties of the model for the space of your artifact. This solution may result in:

Conflict with the model: You may not be able to resolve the conflicts between properties of the space of model and the space of an informative artifact.

To solve this problem, the *Spatial Pattern* implicitly renders the differences between the space of model and the space of an artifact to make the latter free

of micro considerations (sterilized).

Rationale

Most of the nominated models are not navigable and/or interactive and they have a constant and passive nature. If an architectural model is used then the artifact will have a model that is potentially navigable, interactive, changeable, and has a spatial organization.

Resulting Context

Comparing to the other nominated models, the architectural space is the most capable as a spatial model for the informative artifact. The model has a proven user dependency. You can make a movie, create a sculpture or a painting without thinking about your target audience. But you cannot create a building without satisfaction of its users.

Known Uses

Introducing architecture as a reference discipline for Management Information System (MIS). [Lee, 92]

Object-oriented languages use architectural metaphors to explain the whole-parts relationships. [Murdocca, et al., 99]

Computer Architecture uses architectural metaphor of layering to explain its role. [Murdocca, et al., 99]

Related Patterns

Spatial Pattern (of this proto-pattern language).

Sketch

Not applicable.

Examples

Not applicable.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 1: Pattern 2:**

Spatial Pattern

Name

Spatial Pattern

Problem

How should the properties of the abstract spatial model be applied to the space of the informative artifact?

Context

You want to design a navigable and interactive informative artifact. You have applied the ***Spatial Model Pattern*** and you wish to use the architectural space as the spatial model of your artifact. The main spatial properties of the artifact should correspond to the main spatial properties of the model. The eight main properties of an architectural space are as follows: ³⁰

1. **Structure:** The mutual relationship between inner and outer spaces at any level.
2. **Orientation:** The defined directions and leading.

3. ***Sense of place:*** A user can place herself in a defined network of open and filled spaces.
4. ***Continuity:*** Space is a continuous phenomenon combined of open and filled defined spaces relating to each other, creating a path for movement through themselves (a place holder).
5. ***Hierarchy:*** Space is a hierarchy of main (public) and in-fill (private) spaces, as well as open and closed spaces.
6. ***Mental model:*** Space reflects the cultural values of its users and lets them intervene and participate in creating, changing, demolishing, and rebuilding their environment directly or indirectly through the social institutions. Architecture has an inclusive nature. It means that a user has right to build or change her house herself, and the architectural institutions should facilitate this task.
7. ***8. Time-Space:*** Time and space have simultaneous representations. Time adds two properties to the space to create time-space in architecture: ***Movement and Rhythm.***

Forces

Any solution to this problem should balance the following forces:

Differences: Architecture has a real and physical space. A user

physically lives in architectural space. Intellectual enjoyment of living in architectural space is optional depending on the choice of a user. The informative space is an intellectual space and the user cannot live in this space physically.

Similarities: The architectural and informative spaces have some shared values such as, interactivity with the user, and multidirectional navigability. Also, both of these spaces can be designed and organized.

Multiple Definitions / Shared Values: There are different definitions for the architectural space. These definitions have some shared values. You can use only those shared values as the properties of the spatial model for creating an informative space. Those values may be insufficient for defining a space for the artifact.

Solution

Develop a spatial strategy for creating the space of your artifact by defining corresponding - at least - eight main properties to the spatial model (architectural space). Use *Organizational patterns* to implement this strategy.

Rationale

If you define a set of values for creating the space of an artifact, you will have a strategy for your work. The differences between the model and the artifact will

have impact on the level of abstraction in your definitions of those values. Since there is enough similarity, by choosing a proper level of abstraction, you can pass the differences between the space of an informative artifact and the space of its model (simulation). At least, if you define the correspondents for eight main properties of the spatial model with proper level of abstraction, you can be sure that you have the criteria for creating a space. You can reduce the impact of multiple definitions in this way.

Resulting Context

The artifact will have a defined space. Any main property of this space is an abstract interpretation of the correspondent property of the model (architectural space). However, this solution may result in the following problems:

Misinterpretation: Interpretive approach is not reliable if a pure abstraction is not achieved.

Insufficiency: Sufficiency of the model to act as the spatial model of an informative artifact can be supported or denied by time-consuming experience.

Known Uses

Strategic plan of an organization. Where "*organization design begins with the creation of a strategy.*" [Autry, 96]

Related Patterns

Organizational patterns (of this proto-pattern language)

Sketch

Not applicable

Example

Suppose that you want to design an educational CD that contains many different kinds of information. Define a **spatial strategy** at the interaction domain as follows:

Spatial strategy for developing an educational CD.

Preconditions:

- 1- The abstract spatial model of the CD is the pattern based architectural space.
- 2- The content of CD can be classified to some chapters that have a main topic and some supporting documents for some of those main topics. Any part of the CD that shows the content of the main topic will be called "a main node." Any part of CD that shows an independent part of the supporting documents for a main node will be called "a supporting node." The general name for both is "node" as an abbreviation for an informative node.

Goal: Define the spatial properties of the space of the CD.

In this CD, the properties of the space will be as follows:

Structure: Mutual relationships between a node and the CD (the set of inheritance and look up relationships). A node is a miniature of the CD. ⁴⁰

Orientation: The path of navigation to a node that originally starts from the top main node (entry) goes *down* (inheritance). The path of returning from a node that finally ends at the top main node (exit) goes up (look up). The path that relates a node to any other node which participates with the node to support the validity of the content of another node and reveal more information about it, goes horizontal (participation). Therefore, the space of CD has three directions including *Down* (inheritance), *Up* (look up), and *Horizontal* (participation). The reference point of this coordinate system is *Top main node* (entry).

Sense of place: Any node extends the content of a main node to give more detailed information about its content and support the validity of its content. Therefore, each node has a main node. All main nodes are connected to each other through the main path of navigation. The reference point of this local system is *Top main node* (entry).

Continuity: The path of navigation is a continuous path combined of the main path of navigation (that connects all main nodes), and sequential

paths (which connect all participant nodes in a sequence).

Hierarchy: the linkage of any sequential paths to another sequential path is hierarchical and transitional. Transitions will be managed by the main node.

Mental model: The CD is an intellectual product. The user should be able to participate, intervene, and customize the artifact creatively. The CD should reveal information upon request. All cultural values should be respected.

Movement: Navigation.

Rhythm: The CD will have a set of responsibility driven navigational paths with different speed of navigation.

Use ***Organizational patterns*** to implement this strategy.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 2:

Organizational Patterns

This set of patterns refers to the organization of an informative artifact.

N	Problem	Solution	Name	N*
01	What are the possible configurations for organizing the content of an informative artifact?	There are three main possible configurations for organizing the content of an informative artifact that may be combined. Any mixed configuration should have a predominant order. Use the <i>Scalability Pattern</i> to see which configuration should be applied.	Formation Pattern	03
02	What is the impact of size on choosing an order for an informative artifact?	Use simplest order and consider the graded system last. Reorganize the content to reduce the complexity.	Scalability Pattern	04
03	How do you create a graded informative artifact?	Create a sequence of sequences. Use the <i>Sequence Pattern</i> to create any of these sequences.	Grading Pattern	05
04	What are the basic configurations of navigational paths in an informative artifact?	Create a set of distinguished paths including a primary and a secondary subset. Create a dynamic main path of navigation as a part of the primary set of the navigational paths.	Navigation Pattern	06
05	How do you put together the necessary information that can explain an independent part of the content of an informative artifact?	Create a sequence. Manage changes, recovery process, state transitions, and a return path for that sequence. Insert the sequence in the artifact.	Sequence Pattern	07
06	How should a user return to the upper level sequences or exit from a graded informative artifact?	Create a short path independent of the navigational path, by connecting all main nodes of all sequences of the artifact.	Short-Cut Pattern	08
07	How do you manage the transition between and through different levels of a sequential graded informative artifact or the artifact and the background system?	Create a transitional subsystem for your artifact. Manage active grade transitions, passive grade transitions, shortcut transitions, and the impact of changes on the navigational path. Insert the transitional subsystem in the artifact.	Transition Pattern	09
08	What are the default values of an informative artifact?	Apply a low functionality approach for creating default values. Create a small set of elaborate default values with minimum diversity and a small set of elaborated configurations with maximum diversity for each default value and consider those as preferences.	Default values of an informative artifact.	10
09	What is the state of a main node when a user enters to it before or after navigating a sequence of a sequential informative artifact?	Classify states in very general categories. Develop a multiple representation of a main node. Elaborate your artifact by creating an organizational indicator.	State Pattern	11
10	How long should your informative artifact wait for a user interaction before doing the job itself?	Delay in a short-cut path is shorter than delay in a navigational path. Use an organizational indicator in both cases. Test the length of delay with many users.	Delay Pattern	12
11	Can you apply the organizational patterns to an informative artifact with a non-sequential formation?	You can apply the <i>Formation Pattern</i> , the <i>Scalability Pattern</i> , the <i>Delay Pattern</i> , (and in some cases) the <i>State Pattern</i> .	non-Sequential Pattern	13

* The number of a pattern in my proto -pattern language.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 2: Pattern1:**

Formation Pattern

Name

Formation Pattern

Problem

What are the possible configurations for organizing the content of an informative artifact?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact.

The possible formations for organizing the content of an informative artifact in the order of complexity include these three formations: Hub, Linear, Sequential formation, or a combination of them (a mixed configuration). Any mixed configuration may be graded. A graded configuration is more complex than a non-graded one, because it has a multi level organization. Therefore, a mixed configuration combining some sequences with some linear and hub configurations is the most complex formation for an informative artifact. A schema

of these formations is represented in the sketch heading of this pattern.

Forces

Any solution for the problem should balance the following forces:

Scalability: Using a complex order for a simple set or a simple order for a complex artifact is expensive and time consuming. If the order is more complex than is needed, a user will consider it a template. Also, the user will be confused about an informative artifact with a complex and extensive content that has a simple order.

Confusion: The user may be confused about the order of the organization of an artifact that has a mixed order.

Solution

Use the **Scalability Pattern** to see which of these configurations can be applied. If you find out that a mixed configuration should be used, let the artifact has a predominant order. The predominant order of your informative artifact should be the most complex order of the artifact. ⁴¹ For example, if you have a combination of a hub, a linear and a sequential formation in your artifact, the predominant order will be sequential order and this order should be emphasized.

Use **Organizational Patterns** to implement the selected formation.

Rationale

You learn from the abstract spatial model of the artifact (architectural space)

that any order works in the context of a proper scale. The model during its transitional states of the growth, has a predominant order.⁴² The predominant order is the most complex existed order in the architectural space.

Resulting Context

If you emphasize the predominant order of an informative artifact, you make it transparent to the user. Therefore, she is not confused about the order.

Applying the **Scalability Pattern** guarantees that the predominant order of the artifact corresponds to its level of complexity.

Known Uses

Evolution of the city form. [Ardalan, et al., 73]

Related Patterns

Scalability Pattern (of this proto-pattern language)

Sketch

Figure 22: Different possible configurations for organizing the content of an informative artifact (a)

Figure 22: Different possible configurations for organizing the content of an informative artifact (b)

Example

Not Applicable.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 2: Pattern 2:

Scalability Pattern

Name**Scalability Pattern****Problem**

What is the impact of size on choosing an order for an informative artifact?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the *Formation Pattern* and you know that there are many options for organizing the content of your artifact. You want to find a proper formation for your artifact.

Forces

Any solution to the problem should balance the following forces:

Unclear Boundaries: The minimum numerical size of a sequential graded formation is equal to the numbers of the informative nodes that can create

two meaningful sequences (The lower bound limit). The minimum number of informative nodes that can create a sequence is three (otherwise, you will have a hob configuration). Two sequences should share a node to link those together. Therefore, the minimum number of informative nodes that can create a sequential graded formation is five. ⁴³ "Meaningful" is a subjective, content dependent and ambiguous term.

The maximum numerical size of a non-sequential arrangement is equal to the numbers of nodes that can create an accessible ⁴⁴ linear arrangement (the upper bound limit). "Accessible " is a subjective, content dependent and ambiguous term.

Changeability: If the artifact has a sequential graded formation, you can easily expand it to the process of growth without changing the order of the artifact. This means that you can create a new sequence and add it to your artifact. For this reason the non-graded formations have a potential disadvantage. Because when you expand those, the order may change. If you try to create a sequential graded artifact that copes with the potential of changeability, two other forces should be resolved as follows:

Waste: If a graded and expandable artifact did not get a chance to expand, then its potentials are wasted.

User Participation: A user will have a chance to participate in the process of the expansion if an artifact is potentially expandable.

Solution

Use the ***Non-sequential Pattern*** as long as the artifact does not seem too complex. If the artifact is complex reduce its complexity by reorganizing the content of each node. ⁴⁵

Use the ***Sequence Pattern***, to further reorganize the content of an artifact to reduce complexity, if the problem persists. ⁴⁶

Consider the application of graded formation as the last chance. Use a graded formation only if the informative logic of the artifact demands it. Use the ***Sequence Pattern*** to define the sequences. The final step is reorganizing the content of a graded artifact.

This procedure may result in two new problems as follows:

- 1- After all of these efforts, if the complexity problem persists, it means that you have selected an improper medium for representing the content. This conclusion may result from the nature of the medium or its current state of the technology or both. Revise the content.
- 2- The criteria for selecting an order for organizing the content of your artifact will be context dependent and qualitative. Use common sense.

The four other patterns in my proto-pattern language support this pattern to achieve its goals, as follows:

1- To create an open system, the ***Non-sequential Pattern*** includes some techniques for transition from a none-graded system to a graded one.

2- An informative artifact should be expandable to shape a graded system. Therefore, the ***Sequence Pattern*** is developed to be a miniature of the artifact. ⁴⁷

3- To facilitate the creative participation of the user in the process of growth, proper tools are included in the artifact. Use ***Generic Forms Patterns*** to implement these tools. ⁴⁸

Rationale

The spatial model of the artifact has developed different orders to organize its space. The transition from one order to the more complex one in architectural space, is a gradual process balancing a set of forces including resources, external motivations, simultaneous growth and development processes. In architecture, development of and transition to a new order ⁴⁹ is a complex process. The growth imposes the restriction of "accessibility" to the architectural space and creates an internal motivation for changing order. A balance between the "human abilities" and the "natural conditions" determines "accessibility" in

architectural space. Both of these factors can be reinforced by technology in a limited scope. Side effects of the process of reinforcement may intimidate the sustainability of the architectural space. The size or scale is an implicit representation of the "accessibility" factor in the architectural space.

"Accessibility" is a complex factor. ⁶⁰

From the positive point of view, the occurrence of complexity can be considered as a threshold for changing the order. You can use the numerical scientific complexity measures to identify this threshold. ⁶¹

If you want to create an informative artifact that is smaller than a certain size, the application of the numerical scientific complexity measures itself, will be the subject of complexity measurement.

Resulting Context

By following a step by step approach in choosing an order for your artifact, you control the occurrence of complexity. Therefore, most of the time (that this proto-pattern language can be applied to), you do not need to use the numerical scientific complexity measures.

To find the threshold for beginning to measure complexity numerically is out of the scope of this proto-pattern language.

Known Uses

Principles of specialization, outsourcing, and organizational size in organization design. [Tunstall, 99]

Dynamic zoning-matrix in urban planning. ⁵²

Related Patterns

Organizational Patterns (of this proto-pattern language)

Sketch

Figure 23: A linear sequence of informative nodes has no need to use a complex graded system.

Example

Compare the organization of a short story to a novel.

If you want to write a short story you do not divide it into chapters, sections, and parts. If you do, your story seems like a big empty box that contains only a small needle. For a 1000 page novel, you have to use a classification mechanism, otherwise, the reader will not be able to pursue the structure and organization of your novel. A short story can be a chapter of an anthology. An

anthology that does not have a conceptual subtle thread for relating its stories has a destructive nature. As you see, You choose these different formations based of the size of your work.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 2: Pattern 3

Grading Pattern

Name

Grading Pattern

Problem

How can you create a graded informative artifact?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the ***Formation Pattern*** and you know that there are many options for organizing the content of your artifact. You applied the ***Scalability Pattern*** and you are sure that your artifact should have a sequential graded formation.

To create a sequential graded formation you need to classify the content of your artifact. Classification may result in some changes in the content that makes it more detailed or creates some new connections between parts of the content. Content may be ***static*** (predetermined / predefined / template) or ***dynamic***

(changeable). By dynamic content I mean that the final version of your artifact is different from your first draft (a sketch or a scenario) and reflects your findings through processing the draft to the final version (the same as when you process the data to the information). If you use a static content, the first draft and final version should be the same, and you have to undermine your new findings through the process of development of the artifact if they are not in conformity with your predetermined goals.

Forces

Any solution for the problem should balance the following forces:

Ease of Classification: You can classify static content easily by using a top-down approach. Dynamic content can be classified using a top-down / on-the-fly approach. On-the-fly implementation is not a simple task.

Sufficiency, Reuse, Complexity: The process of classification is constructive. If you have a static predefined content, you may find it insufficient, repetitive, or complex, through classification. You can solve the problem of insufficiency, repetitiveness or complexity in a dynamic content on-the-fly and as a part of the process of development of your artifact.

Ease of the revision and reorganization of the content: If you want to reorganize a static content, you should revise the total organization of the

content, and this requires a lot of time. If you have a dynamic content, you can solve the problems on-the-fly and as a part of the process of development of your artifact. You do not need to reorganize the artifact. At the worst case, you should revise only some special cases.

Conformity with the background system: A static content does not have conformity with the dynamic nature of the informative artifact. Dynamic content has conformity.

Chance of user participation: By having static content you may close the door to the user to participate in your artifact. It will lead to a kind of perfectionism. Dynamic content can offer many chances for the user participation.

Solution

If you have a sketch (scenario): Recursively use a top-down approach to divide content to main topics and supporting documents of each main topic. Try to have the minimum number of main nodes that present the main topics. Use a bottom -up approach to determine if the supporting nodes that represent the supporting documents of a topic can create a sequence. Use the ***Sequence Pattern*** to arrange them. Use the ***Non-sequential Pattern*** to arrange those supporting nodes that cannot create a sequence. Add other necessary nodes on-the-fly throughout this process.

If you do not have a sketch (scenario): Use a top-down approach to create a sequence of the main nodes. Recursively create a sequence of the supporting nodes for each main node.

In both cases (with or without having a scenario): Check if the content of some nodes in different sequences can create a shared sequence. Keep the number of these kinds of sequences to as few as possible, because these sequences are a source of complexity. Check if you can reuse any node. Check if the artifact is complex and needs the reorganization of its content. If the artifact seems complex or you can find some repetitive nodes, repeat all above steps. Check all possible options of user intervention to see if any of those options can make the organization of the content complex, to omit them. List the rest of the options to be used as a part of the final list of user participation that will be developed by applying the *User Participation Pattern*.

Rationale

The static content creates a fixed template that has no ability to cope with the changes that will occur due to the constructive nature of the process of classification. Creating a dynamic content may be a complex task. Applying a systemic approach to organize the content can reduce this complexity.

Resulting Context

You apply a step-by-step simple approach to create a dynamic content.

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.⁶²

Your CD may have an introductory part, a descriptive part, and a test part. You applied the ***Formation Pattern*** and the ***Scalability Pattern*** and you are sure that your CD should have a sequential graded formation.

To implement this formation for your CD, create a main node as entry for each of its three parts (introductory, descriptive, and test parts). Use the ***Sequence Pattern*** to create a sequence of these three main nodes.

Each of these main nodes has some supportive documents. Divide those into some nodes based on the logic of your document. If through this process you found out that some supporting documents, which are needed to elaborate the logic of each sequence, are missing prepare them. Create the respective nodes to include these missing parts and add them to the related group of nodes. Use the ***Sequence Pattern*** to create a sequence of each group of these nodes with their own main node.

Your CD has a two level sequential graded formation. One level includes three main nodes of your CD (introductory, descriptive, and test main nodes). At

the second level your CD has three sequences (introductory, descriptive, and test sequences) each containing one of the participant nodes in the first sequence as its main node (introductory, descriptive, or test main nodes).

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 2: Pattern 4

Navigation Pattern

Name

Navigation Pattern

Problem

What are the basic configurations of navigational paths in an informative artifact?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the *Formation Pattern* and you know that there are many options for organizing the content of your artifact. You applied the *Scalability Pattern* and you are sure that your artifact should have a sequential graded formation. You applied the *Grading Pattern* and classified the content of your artifact into some sequences. To connect these sequences and their participant nodes together you may use a set of distinguished paths or a set of similar paths.

Forces

Any solution to the problem should balance the following forces:

Confusion: If you use the similar paths for navigation through your artifact, a user may get confused. She cannot place herself in the space of the artifact, because there is no guidance in the space, and no sense of direction. Distinguished paths can create a sense of direction and place for the user in the space of the artifact.

If you have too many distinguished paths, a user may lose the sense of consistency in your artifact and be confused.

Transparency: Similar paths, do not impose any direction in the space of the artifact. Therefore, the artifact will not have conformity with the background system that basically has direction and consequently cannot make its inner logic transparent to the user. ⁶³

User participation: If you use similar paths for navigation a user can go anywhere in the space of your artifact and no directive restriction will be imposed on her. But she does not know where to go. If you use distinguished paths for navigation, you will impose some directive restriction on the user, but she will know where to go during the navigation.

Solution

Develop a policy for creating a distinguished set of paths including a primary

and a secondary subset.

The primary subset includes: the main path, sequential paths, transitions between sequential paths, and a return path. Use the ***Sequential Pattern*** to implement the sequential paths. Use the ***Transitional Pattern*** to implement transitions and the ***Short-Cut Pattern*** to implement the return path. Any other path will be included in the secondary subset.

At any node, distinguish the connections to the primary and secondary subsets of paths. Do not omit any connection to the nodes of the same sequence without a good reason. Remove any connection from a node to any other main node except the main node of its sequence.

This solution imposes directives for navigation that a user may not like. To solve this problem you can recommend these directives instead of imposing them. It may also produce a large variety of distinguished paths that can confuse a user. To solve this problem you should use a minimum number of variations in your artifact, rendering it free from the small differences.

The ***Default Values of an Informative Artifact Pattern*** supports this solution to face these problems by implementing a low functionality approach for setting the default values of your artifact.

Rationale

If you use similar paths for navigation, it has two negative impacts (user confusion and lack of transparency) that cannot be recovered. If you use distinguished paths, you will face two other negative impacts (user confusion, imposing directives for navigation). You can control these side effects using a low functionality approach in defining the default values of your artifact.

Resulting Context

You show a directed path as a recommended way for navigation through your artifact to the user. She has no obligation to follow this path. You keep the number of various distinguished paths as few as possible. Therefore, users can identify their roles in your artifact easily and without confusion.

Known Uses

Traffic Demand Management (TDM) [HRM, 99]

Related Patterns

Sequential Pattern (of this proto-pattern language),

Transitional Pattern (of this proto-pattern language),

Short-Cut Pattern (of this proto-pattern language),

Default Values of an Informative Artifact Pattern (of this proto-pattern language).

Sketch

Figure 25: A graphic representation of the main path of navigation in a graded Informative artifact.

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact.⁵⁴

You applied the *Formation Pattern* and the *Scalability Pattern* and you are

sure that your CD should have a sequential graded formation.

You applied the ***Grading Pattern*** and classified the content of your CD into three parts including introductory, description, and testing. ⁶⁶ You created a main node and a set of supporting nodes for each part. To connect these nodes together you want to use a set of distinguished navigational paths. To keep the "look and feel" of the CD consistent, you can define a navigational policy for your CD as follows:

Navigation Policy of the Educational CD: At the interaction domain.

Goal: Creating a set of distinguished paths of navigation including primary and secondary subsets for the CD.

The primary subset of navigational paths in the CD includes the following:

1- The main path of navigation which contains a sequential path that relates the three main nodes of the CD (introductory, description, testing) and transitions between them. Any path from a node to any main node, but the main node of its sequence will be removed to preserve the order of navigation in the artifact.

2- Sequential paths contain the three sequential paths that each connecting the supporting nodes of a part of the CD and its main node.

You can call them introductory, description, and testing paths.

3- There are two types of transition in the CD including the followings:

- Transition between introductory, description, and testing main nodes of the CD.**
- Transition between the main or a participant node in any of introductory, description, and testing sequences of the CD and another node in the same sequence.**

4- The return path includes the following portions:

- All short-cuts from any node participating in introductory, description, or testing sequences of the CD to the main node of that sequence.**
- The short-cuts from description, and testing main nodes to the introductory main node.**
- The short-cut from the introductory main node to the outer environment of the CD (which closes the CD).**

The secondary subset of navigational paths includes all paths among all nodes in any of introductory, description, or testing sequences of the CD to another node in the same sequence that is not included in the primary subset of paths. The path from a node to its previous node in any of these sequences should be removed to omit the sense of backward navigation.

The start point of each path in each node will be different in appearance in order to distinguish these paths and visualize this priority rule:

A- The path to the next node in the related sequence (because it is a part of the main navigational path therefore is a member of the primary subset of navigational paths in your CD).

B- The path to the main node of the related sequence (because it is a part of the return path and therefore is a member of the primary subset of navigational paths in your CD).

C- The path to any other participant node in a sequence. (Because it is a member of the secondary subset of navigational paths in your CD).

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 2: Pattern 5**

Sequence Pattern

Name

Sequence Pattern

Problem

How do you put together the necessary information that can explain an independent part of the content of an informative artifact?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the *Formation Pattern* and you knew that there were many options for organizing the content of your artifact. You applied the *Scalability Pattern* and you are sure that your artifact should have a sequential or a sequential graded formation (a sequential formation is the base case of a graded formation).

If your artifact should have a sequential graded formation, you applied the

Grading Pattern and classified the content of your artifact into some sequences.

You can create an ordered sequence or you can give a list to the user in any node to choose her next node without following any order.

Forces

Any solution to the problem should balance the following forces:

Logical Clarity: If you do not use an ordered sequence, the reasoning and logic of putting together that part of the content (that should be included in a sequence) is not obvious for the user. She may get lost, or tired of looking for an inner logic to follow. She may consider the artifact insufficient or very specialized. The first time user will be more disappointed. The less familiar a user with the topic, the more a choice of an informative node will be accidental.

User Participation: If you do not use an ordered sequence, the navigation through your artifact will depend on user choice. Therefore, the potential chance of user participation will be high. When there are a lot of nodes to navigate, the user may not find what she wants and get lost. This will make it impossible to activate all chances of user participation. If you use an ordered sequence, the chance of user participation will be limited.

System Conformity: If you do not use an ordered sequence, the artifact

cannot mirror the inner logic of the background system and has no conformity with its principles.

Upon Request: If you do not use an ordered sequence, the artifact will reveal the information upon user request. However, the search of the content will be difficult and will depend on user experience.

If you use an ordered sequence, the artifact will reveal the information based on its internal logic not the user's requests. If total priority is given to the internal logic of an informative sequence, a user request will be substituted by common sense. In an ordered sequence the search of the content will be very easy and straightforward.

Artifact Integrity: If you do not use an ordered sequence, your approach may have some destructive impacts on the integrity of the artifact, but these impacts are not fatal and are context dependents (some times can be desirable). The nature of an ordered sequential path is constructive, though it is not necessarily dynamic.

Openness: It is easy to add or remove a node to an unordered sequence. There is no order to revise. If you want to add or remove a node from an ordered sequence, you have to check the content of the sequence for the necessary rearrangements.

Solution

Create a sequence: Use an ordered approach ⁸⁸ to create a sequence with a sequential path for navigation as default. Use the *Default Values of an Informative Artifact Pattern* to implement the default value of the path of navigation.

If your artifact has a sequential graded formation, follow these steps:

1. Manage changes: Develop a policy to manage changes upon a user request. Use *Generic Forms Patterns* to implement the necessary tools for informing a user of current formation of the sequential path and to be informed of her requested changes or saving a customized path for navigation.

2. Manage recovery process: Develop a policy to manage recovery of the original default path. Use *Generic Forms Patterns* to implement necessary tools for being informed of a user request for recovery.

3. Manage state transitions: Use the *State Pattern* to manage state transitions of the main nodes when the user enters into or returns from these nodes.

4. Manage the return path: Use the *Short-Cut Pattern* to define a return path from each participant node in the sequence and its related dead ends

to the main node.

5. Insert the sequence in the artifact: Extract any possible option for user participation and add it to the list of user participation options of the *User Participation Pattern*. Extract any possible system request and add it to the list of system requirements.

Rationale

There is no way to omit the destructive side effects of an unordered approach on the reasoning and logic of an arrangement of the content of a sequence. This approach cannot create a transparent view of the background system when it does not follow the same systemic principle. These destructive side effects are inherent in this approach.

The negative side effects of an ordered approach in the user domain can be controlled, though cannot be omitted. You can use a reactive (low functional) ordered system just like natural systems. Low functionality is an approach that is derived from natural processes. Therefore, a user can understand it. This approach increases the chance of user participation. Instead of using a constant path or some predetermined paths (library) for navigation, you can use a dynamic path. It will support the maximum interactivity between a user and the artifact. The visual representation of this dynamic path can make the dynamic nature of the background system and the artifact transparent to the user.

Resulting Context

Your artifact has some ordered dynamic circular sequences whose paths of navigation can be changed on-the-fly by the decision of a user and on possible occasions. The artifact shows the default path and reflects the changes. Creating a dynamic configuration of the sequential path and using minimum functionality approach to create a default path decreased the negative impact of using ordered sequences on the chance of user participation. The order of sequence is a recommendation to a user, not a restriction for her. The artifact will not reveal any information which is not requested by a user.

Some restriction on user participation will be maintained.

Known Uses

Fractal Model of the Life Cycle of Reusable Objects. [Foote, 91] ⁶⁷

Related Patterns

Default Values of an Informative Artifact Pattern (of this proto-pattern language)

Generic Forms Pattern (of this proto-pattern language)

State Pattern (of this proto-pattern language)

Short-Cut Pattern (of this proto-pattern language)

User Participation Pattern (of this proto-pattern language)

Sketch

Figure 26: Options for managing a sequence

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact. ⁵⁸

You applied the *Formation Pattern* and the *Scalability Pattern* and you are

sure that your CD should have a sequential graded formation. You applied the ***Grading Pattern*** and classified the content of your CD into three parts including introductory, description, and testing parts. ⁸⁰ You created a main node and a set of supporting nodes for each part. you applied the ***Navigation Pattern*** to create a set of distinguished paths of navigation in your CD. ⁸⁰ This set includes a primary and a secondary subset of navigational paths. Sequential paths are a part of the primary subset of navigational paths. Your CD has four sequential paths, introductory, description, testing, and main sequential paths.

To create a sequence, you should connect the participant nodes in each of these four sequence through a default circular path of navigation. Any path follows the logic of its related sequence. For example in the introductory sequence, you have four nodes. The main node shows the name and the other information about the author and copyright. The next node contains a description of the content of the CD. The third node contains a description of the prerequisites of the course. The forth node shows a time schedule and necessary tools for following the course (a pen, a brush, a glass of water). A user can navigate these nodes step-by-step and return to the main node. You created a sequence.

A user opens your CD at the main sequence and from there she goes a level down to navigate through the related sequence of a main node by selecting it.

She should come back to the main sequence to go to any other sequence. Your CD has a two level sequential graded formation.

If you can find some supporting document for any of the supporting nodes in one of the three existing sequences that can create a sequence with that node, you should develop a new sequence and your CD will have a layer more. This is similar with the idea of "a node within a node, within a node ..." [Foote, 91] or a fractal configuration.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 2: Pattern 6:**

Short-Cut Pattern

Name

Short-Cut Pattern

Problem

In a graded informative artifact, how should a user return to the upper level sequences or exit from the artifact?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the ***Formation Pattern*** and you know that there are many options for organizing the content of your artifact. You applied the ***Scalability Pattern*** and you are sure that your artifact should have a sequential graded formation.

You applied the ***Grading Pattern*** and classified the content of your artifact into some sequences. You applied the ***Navigation Pattern*** and developed a

policy to create primary and secondary sets of navigation paths. You want to create a return path as a part of the primary subset of navigational paths.

This scenario shows the worst case in creating a return path in an informative graded artifact: "a user navigating the deepest dead end of a nested sequence in a graded artifact, wants to return to the one of the previous sequences, main menu or close the application. She is not patient to return from the same path that she has used to go there. If there is not a short cut, she should use the system tool to close the application. Then she has to restart the application and navigate to reach the targeted node."

You can have a path to return to the entry node of your artifact from each node (go-to approach), or you can have a path to return to the main node of the related sequence and from there step by step to the main node of each upper sequence (ordered approach). In an ordered approach, the navigation path may work as a return path or you can define a short-cut return path independent of the navigation path.

Forces

Any solution for this problem should balance the following forces:

Sense of place: If you use a go-to approach, when a user returns from a deeper sequence to the entry node without passing the upper sequences,

she cannot find her place in the space of the artifact. In the same situation, if you use an ordered approach the user can easily locate herself in the space of the artifact, because she has passed through the hierarchy of the artifact.

User Participation: If you use a go-to approach, any navigation through the space of the artifact is permitted by the decision of the user. But when she decides to close the artifact, if she changes her decision, she has no other way but restarting the artifact. Going back to the first location (undo) and trying again, or changing her path after arriving at the new location do not help her in this situation. Displaying a save message is helpful but it stops the process at the highest level (the main node). If nothing should be saved, the process won't be stopped.

If you use an ordered approach, not any navigation in the space of the artifact is permitted. But a user has enough time in any sequence to change her decision to leave a sequence or closing the artifact.

System Conformity: The background system of the artifact follows an ordered configuration, therefore an ordered approach makes the logic of the background system transparent to the user, a go-to approach does not.

Upon Request: If you use a go-to approach, the return path will be

activated upon user request. The activation of the return path is as fast as requested. But it cannot be deactivated if the user changes her decision. If you use an ordered approach, the return path will be activated upon user request and can be deactivated if user changes her decision. However, there are some options, that if requested by the user, will be denied.

Time Consuming: If you use a go-to approach, a user can pass the return path very quickly. From a user's viewpoint an ordered approach that uses the navigation path as a return path is time consuming. But, an ordered approach that defines a short-cut as the return path is not.

Solution

Create a short path independent of the navigational path to return from any node to the main menu, connecting all main nodes of all sequences participating in the navigation path.

Do not create any return path from a node to the other main nodes but the main node of its sequence.

Use the ***Delay Pattern*** to activate the return path by the artifact if a user does not perform the necessary action at a certain time.

Use the ***Transition Pattern*** to implement the transitions between a lower level and an upper-level main node through the return path.

Use the **State Pattern** to implement the state of a main node through the return path.

Rationale

A user is patient to navigate an artifact, but she does not like to waste a lot of time to return from deep inside of a graded artifact. Therefore, the return path should be shorter than the navigational path.

By using an ordered approach, the user is not free to go anywhere in the space of your artifact (as she is in unordered approach), but she has more control on the impacts of her choices or decisions on the artifact.

Resulting Context

By applying this pattern the return path to the upper sequences and out of the artifact is short. This path is not as short as a return path in a go-to approach, but it is faster than using the navigation path as the return path.

A user can change her decision, at any main node (a step). She has complete control (not complete freedom) on the results of her choices or decisions. By applying the **Delay Pattern** a user can wait and watch the artifact do the job for her.

Known Uses

The network of pedestrian movement inside the network of a city transportation system, where priority is given to the pedestrian movement.

[Spacesyntax, 99]

Figure 28: A profile of the short-cut path in an informative artifact with a graded formation

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact.⁶¹

You applied the *Formation Pattern* and the *Scalability Pattern* and you are sure that your CD should have a sequential graded formation. You applied the *Grading Pattern* and classified the content of your CD into three parts including introductory, description, and testing parts.⁶² You created a main node and a set of supporting nodes for each part. you applied the *Navigation Pattern* to create a set of distinguished paths of navigation in your CD.⁶³ This set includes a primary and a secondary subset of navigational paths. The return path is a part of the primary subset of navigational paths.

A user passes through this path in your CD to return from any of three lower sequences (introductory, descriptive, and testing sequences) to the main sequence or to go out of (closing) the CD environment. You should connect all participant nodes of each of the three sequences to the main node of their sequence (not to any other main node). The main node of the main sequence (introductory) is responsible for closing the application through a look up transition. This transition should be as simple as possible because complicated transitions are time consuming and are not desirable in a return path. You also need to display a save message before applying this transition. You can use the *Transition Pattern* to implement this transition.

Since you applied the *Navigation Pattern* you know that the priority of the opening to the return path in any node is less than the opening to the main navigational path. Although, this opening has priority to any other opening. This priority rule visualizes the possibility of using the return path and indicates that this is not a recommended way for navigation through your CD.

Use the *State Pattern* to show the state of any of the three main nodes through the returning path. A main node may be navigated (all nodes of its related sequence are visited), visited (not all nodes of its related sequence are visited), or unvisited (no nodes of its related sequence are visited). Returning from the inner sequences of a sequential graded formation normally needs more

than one "click." In your CD the number of clicks for returning from a node in any of the three lower sequences and closing the application is two. For this reason you may want to have a more sophisticated return path. You can use a timer to activate transition to the upper sequence from a lower one (for example from the description sequence to the main sequence), if the user does not interact with the CD during a certain period of time.

Using the *Delay Pattern* guarantees that a user has enough time to interact with the CD before the CD activates the transition itself. Also, the timer can be activated when a user decides to close the CD using the system tools (such as Ctrl+Alt+Del). The timer can push her through the return path, displaying the save message and then closing the CD.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 2: Pattern 7:**

Transition Pattern

Name**Transition Pattern****Problem**

How do you manage the transition between and through different levels of a sequential graded informative artifact, or the artifact and the background system?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the ***Formation Pattern*** and you know that there are many options for organizing the content of your artifact. You applied the ***Scalability Pattern*** and you are sure that your artifact should have a sequential graded formation.

You applied the ***Grading Pattern*** and classified the content of your artifact into some sequences. You applied the ***Navigation Pattern*** and developed a policy to create primary and secondary sets of navigation paths. You applied the ***Sequential Pattern*** and the ***Short-Cut Pattern*** to create a navigational and a return path as a part of the primary subset of navigational paths.

To manage the transition between different kinds of nodes through different paths of navigation, or the artifact and the background system you can use an *ordered* or an *unordered approach*. In an unordered approach you can use any kind of transition between two nodes regardless of their organizational role in the artifact. In other words, you can design the transitions according to the requirements of a part of the content (cinematographic approach) ⁶⁴ or your taste.

Forces

Any solution to the problem should balance the following forces:

Sense of Place: If you have an ordered set of transitions in a graded artifact, it will give a sense of consistency to the user that helps her to locate herself in the space of the artifact. For example, if you use a cover-up transition in all transitions between two main nodes, as she sees a cover-up transition in the artifact she knows that she is transferred to another main node. If you do not use an ordered approach, after transition

from one node to the other one, she cannot find the role of the new node in your artifact and will be confused about her place in the space of your artifact.

Integrity and Continuity: Transitions sew together the different nodes of your artifact. If you use an ordered approach for creating transitions, it will help to create a sense of continuity and integrity in your artifact. If you apply a transition according to your taste, or the content of a node, and you do not use that in all similar cases consistently throughout your artifact, that transition cannot sew the parts of your artifact together. These kinds of transitions are decorative or symbolic and are not constructive.

Indicative Value: You may believe that if you repeat a well-designed transition between two nodes, the importance or the indicative value of that transition will be ignored.

User Participation: If you use a predetermined and unchangeable transition in an ordered or unordered approach, you will reduce the chance of user participation. As a result of applying the *Navigational Pattern*, a user can change the inner sequential navigation path. If you relate any transition to a special node or part of the content (cinematographic approach) the change in the navigation path may have conflict with your transitional sub-system.

Solution

Use an ordered approach with minimal functionality to develop a transitional sub-system by following these steps:

1- Manage active grade transitions: Identify all meaningful options of grade transitions for a user as *Active Grade Transitions*. Also identify those active grade transitions that are not permitted in all cases as *protected Grade transitions*. You should consider the latter group as the special cases.

2- Manage passive grade transitions: Identify all unacceptable options of grade transitions from system viewpoints. You should remove these options.

3- Manage shortcut transitions: Identify all short-cut transitions, including those that may occur during inter-sequential navigation or in a short-cut path.

4- Manage the impact of changes on the navigational path: Do not relate the transition to a special node or part of the content. If you want to have these transitions, do not consider those transitions as a part of your transitional sub-system. The transitions in your transitional sub-system are dependent on the position of a node in the organization of your artifact.

Any change in the navigational path by the user is inter-sequential and does not have any impact on your transitional sub-system.

5- Insert the transitional sub-system in the artifact: Extract any possible option for user participation and add it to the list of user participation options of the *User participation Pattern*. Also extract any possible system request and add it to the *list of system requests*.

Use *Generic Forms Patterns* to implement any of these transitions.

Rationale

If you apply transitions only in those cases that really need them, you can decrease the number of transitions. Therefore, by indicating the importance of a transition in your artifact even though, the number of times that any transition should be repeated is the same, the indicative feature of a repetitive transition is promoted.

An informative artifact is not a cinematographic product that uses content dependent transitions in all sequences. If you want to use these kinds of transitions you can consider them as separate cases.

An organizational transition is related to the position of a node in the organization of the artifact, not to the content of that node or the taste of the designer. You can increase the chance of user participation if you let a user

customize the look and face of the transitions, or disable them.

Resulting Context:

The resulting ordered sub-system helps a user to locate herself in the space of the artifact. For example, suppose that she moves from a main node of a sequence to a main node of another sequence that has the same position (level) in the space of the artifact. The next time that she moves between two nodes, which have the same status, because of the similarity of transitions, she will understand that this transition is horizontal and both nodes have the same position in the organization of the artifact.

You promote the importance of any transition in your artifact, regardless of the number of times that it should be repeated. You do not impose any restriction on the user because of your taste, artistic tendencies, or technological magic.

Known Uses

Patterns of transition between different levels of the transportation system in a populated city. According to these patterns you cannot enter from a small local lane to a highway. [HRM, 99]

Related Patterns

Generic Forms Patterns (of this proto-pattern language),

User Participation Pattern (of this proto-pattern language).

Sketch

Figure 29: A graphic representation of transitions between different nodes in an informative artifact with a graded formation

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact.⁶⁵

You applied the ***Formation Pattern*** and the ***Scalability Pattern*** and you are sure that your CD should have a sequential graded formation. You applied the ***Grading Pattern*** and classified the content of your CD into three parts including introductory, description, and testing parts.⁶⁶ You created a main node and a set of supporting nodes for each part. you applied the ***Navigation Pattern*** to create a set of distinguished paths of navigation in your CD.⁶⁷ This set includes a primary and a secondary subset of the navigational paths. The return path is a part of the primary subset of navigational paths. You applied the ***Sequential Pattern*** to implement the main navigational path⁶⁸ and the ***Short-Cut Pattern*** to implement a return path⁶⁹ for your CD.

If you want to develop a ***Transition sub-system*** for your CD to manage the transitions among all kinds of nodes and from the entry node (the introductory main node) to the system, follow these steps:

- 1- For Managing active grade transitions make a table including all options of ***meaningful grade transitions for the user*** in your CD (***Active Grade Transitions***). This table is similar to the Table 8. Table 8 shows that you need to design three kinds of active grade transitions including ***Look up, Vertical, and Horizontal***, each of them may appear in different levels. The transitions that are listed in the first three rows of Table 8 are system driven and defined by the system. You do not need to implement those

transitions. Though you need to know the special features of the system tools, to let the system invoke the save message of the CD. Your CD may ask a user to answer some questions as part of its educational course, and she wants to save her answers somewhere before closing the CD.

The save message will let her to do that.

From:	To:	Type	Direction	Responsibility of:	Restriction
-	System	Log-in	-	System	Potential
System	-	Log-out	-	System	Hybrid (Save message or Single Inheritance)
System	CD (Entry) Main node	Enter	Vertical	System	Potential
CD (Entry) Main node	System	Exit	Look-up	Application/ Main Node	Hybrid (Save message or Single Inheritance)
Main Node	Main Node	Transition	Horizontal	Main Node	Hybrid (None or Protected) *
Main Node	Sequence	Enter	Vertical	Main Node	Single Inheritance
Sequence	Main Node	Return	Look-up	Sequence	Single Inheritance
Sequence A	Sequence B	Transition	Horizontal	Sequence	Protected
Sequence	Dead End	Enter	Vertical	Sequence	Single Inheritance
Dead End	Sequence	Return	Look-up	Dead End	Single Inheritance
* A hierarchy of entries will impose a protected restriction on transition between main nodes. Otherwise there will be only one node and there is no restriction.					

Table 8: Active Grade Transitions in an informative artifact (Educational CD)

By imposing a single inheritance restriction it is impossible to move from a supporting node in a sequence (for example the introductory sequence) of your CD to a supporting node of any other two sequences. Therefore, no direct transition from a node in a sequence to a node of the other sequence is permitted (multi-sequential horizontal transitions). Creating

many of these kinds of paths results in chaos in the organization of the artifact and consequently in the mental model of its user. If you have a good reason, you can create a multi-sequential sequence (including some nodes of different sequences) with a multi-sequential horizontal transition. For a very special reason, you may want to connect a supporting node in the descriptive sequence of your CD to two nodes in its testing sequence (one of the testing nodes shows a test and another one shows its answer). This may be done, for example, if the result of the test is a prerequisite for understanding the content of a node in descriptive sequence. This sequence is a special sequence. Any transition to and from this special sequence is only permitted to a sequence that the main node of this special sequence is a participant of that. This means that in your CD you cannot enter to the sequence that is created by connecting a descriptive node and two testing nodes, or transit from this sequence to the testing sequence. ⁷⁰

You can apply the *User Participation Pattern* to let a user customize any of the active grade transitions. Though, you should not let a user remove these transitions. This is the only restriction that you should impose on a user's choice.

To visualize the different organizational levels of your CD that these

transitions pass through do not use very different types of transitions. If a user can distinguish the different levels of organization of your CD by passing through these transitions, it means that these transitions work.

2- Use Table 8 to manage passive grade transitions. Table 8 shows that only 10 of 23 possible options for moving between five participated environments in this table ⁷¹ are active transitions. There are eight transitional options that the system does not let you to use including the following cases:

- If a user is logged-off, the only transitional option is to log in, this omits three options.
- If a user logs-in she can only log-off or open the CD, this omits two other options.
- If a user is in the CD's environment, she cannot log off using CD (application) tools. This will omit three other options.

These eight options are automatically removed by the system, as a part of its systemic protection layer.

The five other options can be activated or disabled according to a user's (or a user interface designer's) decision. Give the priority of decision making to the user of your CD in order to increase the chance of user

participation. These options include:

- Exiting from a sequence or dead end to the system. The system can automatically close the CD (two options),
- Returning from a dead end to the main path or vice versa (two options),
- Transition between dead ends (one option.)

Remove all of these options from the default configuration of your CD. But give an opportunity to the user for changing the default values and playing with these options. This policy is an application of the minimum functionality approach that gives maximum priority to clarity of the navigation through your CD. The original configuration of your CD should be immediately restored upon user request.

3- To manage short-cut transitions you should implement the other two transitional options that short-cuts can create between supporting nodes and the main node of each of three sequences of your CD, or the three main nodes (introductory, descriptive, and testing main nodes) that participate in the short-cut path.

4- At this step you should manage the impact of changes in the navigational path. If a user makes a decision to enter a node from a path

other than the navigational path (just like when she enters a house from the side door) she should feel that it is not the main route.

In your CD a user may want to pass from a node in your testing part and go to the next node because she does not want to work on a special test. She should be able to do that, but she should understand that this is not a recommended path.

Use ***Generic Forms Patterns*** to implement the transitions. Keep these kinds of transitions as simple as possible.

All of these transitions (except the protected transitions) are independent of the content. Otherwise, if a user changes the navigation path and saves a new path for the navigation, the transitional sub-system of the CD cannot work.

5- To insert the transitional sub-system in the artifact, make a list of all ***Active Grade Transitions*** and ***Passive Grade Transitions*** that are acceptable to the system (thirteen options) in your CD. Add this list to the user participation list of the ***User Participation Pattern***.

The list of system requirements of your CD may include some requirements for running your CD, some memory request for saving the result of tests, and a request for invoking "save message" command of the

CD (when it is open and a user wants to log off.) These requests are subject to your negotiation with system designers.

Chapter 4: Section 3: Part 2: Pattern 8

Default Values of an Informative Artifact

Name

Default Values of an Informative Artifact.

Problem

What are the default values of an informative artifact?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the ***Formation Pattern*** and you know that there are many options for organizing the content of your artifact. You applied the ***Scalability Pattern*** and you are sure that your artifact should have a sequential graded formation.

You applied the ***Grading Pattern*** and classified the content of your artifact into some sequences. You applied the ***Navigation Pattern*** and developed a

policy to create primary and secondary sublets of navigational paths. You applied the ***Sequential Pattern*** and the ***Short-Cut Pattern*** to create a navigational and a return path as a part of the primary subset of navigational paths.

To manage the transition between different kinds of nodes through different paths of navigation, or the artifact and the background system you applied the ***Transition Pattern*** and developed a transitional sub-system.

To realize the strategy, policies, and the transitional sub-system that you implemented, you need to have a policy for determining the default values of your artifact. You can follow a maximum functionality approach, or a minimum functionality approach to implement the default values

Forces

Any solution to this problem should balance the following forces:

Elaboration: If you use a maximum functionality approach to define the default values of your artifact, it will have an elaborate final status. If you use a minimum functionality approach to define those values, the nature of your artifact will be temporal.

Changeability: Changing an artifact which has a final elaborate status is difficult, but if the artifact has a temporal status it will cope with the changes easily.

User participation: If you try to keep the elaborated status of an artifact untouched, the user will be an audience for, not a participant to the artifact. A user can actively participate in reshaping of an artifact with a temporal nature.

Organizational logic: The elaborated status supports organizational logic. But excessive elaboration can work against organizational logic and create a formalistic view of the organization.⁷² The Organizational logic may be undermined when you try to implement a minimum functionality approach.

Cost (Time and Money): By using a minimum functionality approach for developing the default values of your artifact you decrease the production cost (because more time and money are needed for implementation of an elaborated artifact), you reduce the revision cost (because implementation of a revised version of an elaborated artifact is more expensive), and you reduce the cost of maintenance of the artifact (An elaborated artifact cannot cope with the rapid technological changes, when it tries to keep its elaborated status untouched. Therefore, you should develop a new version of your artifact very soon.)

Solution

Follow a minimum functionality approach for creating default values for your

artifact. Follow these steps to implement this approach:

- 1- Create a small set of elaborate default values with minimum diversity, and a small set of elaborated configurations with maximum diversity for each default value and consider those as preferences.
 - 2- Identify all configurations and default values that may be destructive to the organization of your artifact, and remove them. Consider any other possible configurations and values of your artifact customizable.
 - 3- Check any negative side effect of applying the two previous steps on the minimum functionality approach, and organizational logic of your artifact. Recursively apply this solution to eliminate these side effects.
- Use *Delimiter Patterns* and *Generic Forms Patterns* to implement the default values.

Rationale

The artifact should make the principles of the background system transparent to the user. Transparency implicitly supports the elaboration of the artifact because the background system is elaborated. Therefore, only values and formations should be elaborated.

You should apply a minimum functionality approach to support the user

participation principle. Therefore, the negative side effects of elaboration on the minimum functionality approach should be checked and omitted.

Resulting Context

You limit the process of elaboration to values and formations, and recursively preserve the organization and main principle (minimum functionality approach) of your artifact, through a recursive checking mechanism.

Known Uses

Frames [Lemon, et al., 99], ⁷³

Adapter classes in Java. ⁷⁴

Related Patterns

Delimiter Patterns (of this proto-pattern language)

Generic Forms Patterns (of this proto-pattern language)

Sketch

Not applicable.

Example

Adapter classes in Java

For omitting multiple inheritance to gain security, Java divides abstract classes with maximum and minimum functionality as abstract class and interface,

and applies different restrictions on each of those as follows:

1- A user cannot extend her defined class from more than one abstract class (that has maximum functionality). But she can implement multiple interfaces (that has minimum functionality) for a class. Java does not impose the restriction of multiple inheritances on pure abstract classes (interfaces) because it is not dangerous from a security point of view.

2- A user can use an abstract class instead of an interface. But she cannot use an interface instead of an abstract class. This is rational, because a pure abstract class cannot have any concrete method.

3- If a user wants to implement an interface or an abstract class, she should implement all its abstract methods because those are abstract.

These three simple and rational rules show that Java uses a minimum functionality approach to implement its strategy for facing the issue of abstract classes and multiple inheritance.

Java follows the same approach in implementing Package `java.awt.event`.

There are 11 interfaces in this package, each containing one to seven abstract methods. The worst case is *WindowListener* interface with seven methods. The implementation of all of these methods is too much work for a user. A user may forget to do that. This was a side effect of applying Java's main

strategy. Therefore, Java creates an abstract class, with a formal (not actual) maximum functionality (adapter abstract class) to set a minimum functional default value (empty body = do nothing) for each abstract method. This is too much work for an interface with only one abstract method. Therefore, Java does not implement any adapter for these kinds of interfaces.

In this case java uses a low functionality approach to implement its strategy for facing the issue of interfaces with many methods. Also, there are some unsolved side effects as follows:

- 1- If a user has chosen to implement a minimum functionality abstract class instead of an interface she will be overburdened. She would not be able to use all facilities of the system and should define as many methods as requested in that abstract class.
- 2- A user can declare static variables in abstract class and interface. She can declare a variable in an abstract super class and in an interface by the same name. If she tries to use that variable inside any class that extends and implements that abstract class and interface, she will get an error message. This means that Java distinguishes interfaces from abstract classes only by keywords not by the type of pointers. In other words, for Java a pointer to the abstract super class and a pointer to the interface have the same priority.

There exist two other side effects that the reasons for which are not clear (to me):

1- If a user defines an abstract class with no concrete method Java does not prompt an error message to inform her that it is an interface.

2- Considering Interface as a storage for static variables and having two pointers with the same priority, one to the abstract class and one to the interface does not support the organizational logic of Java.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 2: Pattern 9:**

State Pattern

Name

State Pattern ⁷⁵

Problem

What is the state of a main node, when a user enters to it before or after navigating a sequence of a sequential graded informative artifact?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model for your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the ***Formation Pattern*** and you know that there are many options for organizing the content of your artifact. You applied the ***Scalability Pattern*** and you are sure that your artifact should have a sequential graded formation.

You applied the ***Grading Pattern*** and classified the content of your artifact into some sequences. You applied the ***Navigation Pattern*** and developed a policy to create primary and secondary sublets of navigational paths. You applied the ***Sequential Pattern*** and the ***Short-Cut Pattern*** to create a navigational and a return path as a part of the primary subset of navigational paths.

By navigating these paths you pass through main nodes. You may or may not want to indicate the state of a main node to show if all, some, or none of its supporting nodes are navigated.

Forces

Any solution to the problem should balance the following forces:

User Confusion: If you do not indicate the state of a main node through navigation, a user that enters that main node will be confused. She cannot find out if she visited all (navigated), some, or none of the nodes participating in the sequence of that main node.

Display Consuming: You can rescue a user from confusion if you have an organizational indicator in any participant node of a sequence, and on that indicator highlight the nodes that are visited. You may use a set of widgets (links, buttons, . . .) instead of an indicator (just like many web pages that are currently on the web.) Having this indicator or set of

widgets in all nodes is display consuming and its repetition may make a user tired.

Upon Request: if you have an organizational indicator or a set of widgets in any participant node of a sequence, they are always there whether a user requests to see them or not.

Solution

By classifying states in very general categories keep the number of states minimum (You do not need to show whether one supporting node or two supporting nodes of a main node are visited.)

Develop a multiple representation of a main node to show multiple states of its related sequence including: Visited (not all nodes of the sequence are visited), Navigated (all nodes of the sequence are visited) and unvisited (none of the nodes of the sequence is visited).

Elaborate your artifact by creating an organizational indicator or a drop down menu that contains some widgets in the main node. The indicator or the menu will be displayed only if a user requests and shows references to the participant nodes in the sequence and their state (visited or unvisited).

Use **Generic Forms Patterns** to implement the indicator or the drop down menu.

Rationale

You can consider the state of a sequence as the property of its main node, and a main node as an object that is responsible for representing its state by its own look and feel.

If a user wants to see the content of the sequence, or to know which nodes are visited, she can open the organizational indicator (or a drop down menu including some widgets). This is just like sending a message to an object to get some detailed information about one of its known properties.

Resulting Context:

By entering a main node a user knows the state of its sequence. To have detailed information about the state of the sequence she can open the indicator or drop down the menu. You need only one widget to open the indicator or drop down menu in the main node. If a user who navigates a supporting node of the sequence wants to have detailed information about the state of the sequence, she can use the short-cut path to return to the main node of sequence which is responsible for representing the information about the state. Therefore, this information is displayed upon a user request and the limited surface of the display will be saved.

Known Uses

State Pattern [Gamma, et al., 95]

Related Patterns

Generic Forms Pattern (of this proto-pattern language)

Sketch

Not applicable.

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact. ⁷⁶

You applied the ***Formation Pattern*** and the ***Scalability Pattern*** and you are sure that your CD should have a sequential graded formation. You applied the ***Grading Pattern*** and classified the content of your CD into three parts including introductory, description, and testing parts. ⁷⁷ You created a main node and a set of supporting nodes for each part. you applied the ***Navigation Pattern*** to create a set of distinguished paths of navigation in your CD. ⁷⁸ This set includes a primary and a secondary subsets of navigational paths. The return path is a part of the primary subset of navigational paths. You applied the ***Sequential Pattern*** to implement the main navigational path ⁷⁹ and the ***Short-Cut Pattern*** to implement a return path ⁸⁰ for your CD.

You want to show the states of any main nodes to a user that navigate the CD through these paths. You can classify these states to: " unvisited, visited, and navigated states." To show these states you can choose three different background colors (for example a tonality of blue from light blue to dark blue.)

If a user enters to a main node for the first time, for example to the testing

main node of your CD, the background of this node is light blue. She can click on a widget ("Show Sequence" button, for example) to open an indicator that shows the organization of the testing sequence.

If she navigates two nodes of the testing sequence and returns to the testing main node, the background of this node is medium blue. If she clicks on "Show Sequence" button, she can see that any reference to those visited nodes is highlighted.

If a user navigates all nodes of the testing sequence and returns to the testing main node of the CD, the background of this node is dark blue. If she clicks on "Show Sequence" button, she can see that no reference is highlighted.

Use ***Generic Forms Pattern*** to implement a transparent indicator.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 2: Pattern 10**

Delay Pattern

Name

Delay Pattern

Problem

How long should your informative artifact wait for the reaction of a user before doing the job itself?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the ***Formation Pattern*** and you know that there are many options for organizing the content of your artifact. You applied the ***Scalability Pattern*** and you are sure that your artifact should have a sequential graded formation.

You applied the ***Grading Pattern*** and classified the content of your artifact into some sequences. You applied the ***Navigation Pattern*** and developed a

policy to create the primary and the secondary sub-sets of navigational paths.

You applied the ***Sequential Pattern*** and the ***Short-Cut Pattern*** to create a navigational and a return path as a part of the primary subset of navigational paths.

A user may forget to interact with the artifact through a short-cut path or a navigational path to direct the flow of its motion. If her delay exceeds a certain amount of time, your artifact can do this job by itself or can do nothing.

Forces

Any solution to the problem should balance the following forces:

User Participation: If the length of delay for reactivating a navigational or a short-cut path by the system is equal to the length of time that a user needs to pass through that step of the path by herself, it means that a user action or decision has no impact on the flow of display of your artifact (just like a movie). Therefore, the chance of user participation and her interaction with your artifact will decrease.

User Confusion: When a deactivated navigational or short-cut path is reactivated by the artifact a user may be confused and unable to locate herself in the space of the artifact. She losses her control on the artifact because of the system intervention.

Differences: The navigational path and the short-cut path have different rhythms of navigation. A user passes a navigational path much slower than a short-cut path.

In a short-cut path the destination of reactivation of the path by the artifact is its next step that finally closes the artifact, after saving a back-up of changes. In a navigational path you have different choices in determining the next or the final destinations.

Solution

Delay through the short-cut path should be as short as a user has enough time to change her decision. But delay through the navigational path should be as long as a user has enough time to read the content of the node that she is in, and make a decision to choose a path.

Create an indicator to be displayed and shows the previous steps of a navigational or short-cut path if artifact pushes itself through that path. The indicator will be removed at any step that a user reacts to the artifact. This indicator can fill the display to work as the next destination for the navigational path. The final destination in both cases is closing the artifact. Use the **Generic Forms Pattern** to implement this indicator.

Test the length of delay in both cases with different type of users, specially

with a large number of its potential end users. ⁸¹ Apply the *User Participation Pattern* to indicate on the diversity of users of your artifact, and not to generalize the users as a common type (Human Person).

Rationale

If you reduce the length of delay through the short-cut path, the impact of "multiple clicks" will be reduced. ⁸²

If you increase the length of delay through the navigational path, a user does not feel that you want to push her through the artifact.

You can add an independent node to the artifact to work as a destination for artifact through the navigational path.

Resulting Context

A user can locate herself in the space of the artifact if she left it for any reason (opening another application or having a glass of water) by the help of the indicator.

A user is not pushed through the navigational path because of her delay in interacting with the artifact.

Known Uses

Pedestrian lights in a Traffic light system.

Related Patterns

Generic Forms Patterns (of this proto-pattern language)

User Participation Pattern (of this proto-pattern language)

Sketch

Not applicable.

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the **Spatial Pattern** to develop a spatial strategy for defining the properties of the space of your artifact. ⁸³

You applied the **Formation Pattern** and the **Scalability Pattern** and you are sure that your CD should have a sequential graded formation. You applied the **Grading Pattern** and classified the content of your CD into three parts including introductory, description, and testing parts. ⁸⁴ You created a main node and a set of supporting nodes for each part. you applied the **Navigation Pattern** to create a set of distinguished paths of navigation in your CD. ⁸⁵ This set includes a primary and a secondary subset of navigational paths. The return path is a part of a primary subset of navigational paths. You applied **Sequential Pattern** to implement the main navigational path ⁸⁶ and **Short-Cut Pattern** to implement a return path ⁸⁷ for your CD.

If a user wants to open her mail box and then return to the CD, she may not remember her navigated parts of your CD. You can create a general node to be displayed after a certain time and which contains an organizational indicator that shows her the navigated sequences and nodes.

If a user does not want to use multiple clicks to go to the upper sequences or to close the CD, she can delay for a reasonable time and the CD will do the job for her. The length of delay in both cases is dependent on the content of the main nodes in your CD and the maximum amount of information that are included there. Test your CD by different users to check if the length of delay in navigational and short-cut paths is convenient for them.

Apply the ***User Participation Pattern*** to indicate on the diversity of users of the CD. Use ***Generic Forms Pattern*** to implement the indicator.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 2: Pattern 11**

None-Sequential Pattern

Name

Delay Pattern

Problem

Can you apply the **Organizational Patterns** to an informative artifact with a non -sequential formation?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the **Spatial Pattern** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the **Formation Pattern** and you know that there are many options for organizing the content of your artifact. You applied the **Scalability Pattern** and you are sure that your artifact should have a none sequential formation (a hub or a linear formation or a combination of those.)

Forces

Any solution to the problem should balance the following forces:

Application of other organizational patterns: Non-sequential formations are more simple than sequential one. You do not need to apply all but some of those may be applicable.

Solution

You need to apply the **State Pattern** to a hub configuration and the **Delay Pattern** to the hub and linear configurations. Both of these patterns should be applied to a combination of those. Use a very simple modification of these patterns.

Rationale

A hub is an informative node whose supporting nodes are all dead ends. Therefore, potentially there is a return path in a hub that always returns back to the main node of the hub. This implies that the State Pattern is applicable in this formation. A linear formation does not have a return path. Therefore, you cannot apply the State Pattern there.

In any non-sequential formation a user may forget to interact with the artifact. Therefore, the Delay Pattern can be applied in all cases.

Resulting Context

You can apply maximum four patterns of ten Organizational Patterns in your artifact if it has a non-sequential formation (Formation, Scalability, Delay, and

State Patterns.) The rest of these ten patterns cannot be applied because these formations are simpler than sequential formation.

Known Uses

Not yet.

Related Patterns

Delay Pattern (of this proto-pattern language)

State Pattern (of this proto-pattern language)

Sketch

Figure 30: A graphical representation of non-sequential formations of an informative artifact.

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model

of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact. ⁸⁸

You applied the ***Formation Pattern*** and the ***Scalability Pattern*** and you are sure that your CD should have a none-sequential formation. You classified the content of your CD into three parts including introductory, description, and testing parts. You created a main node and a set of supporting nodes for each part that are independent of each other. It means that you cannot create a sequence of those.

For example, your description main node has three independent supporting nodes (description of course in English, description of course in French, description of course in Spanish.) It is obvious that they cannot create a sequence, because there is no logical organizational path between them.) A user in the description main node can choose one of those and go there. Later, she can come back to the main node (therefore you should apply State Pattern) and should find it in a new state. If a user does not interact with the supporting node (for example English Version of the course) during a certain amount of time, CD will display an organizational indicator to show her the whole organization of the CD, and the nodes that she visited will be highlighted (therefore you should apply Delay Pattern).

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 3:

Delimiter Patterns

This set of patterns refers to the delimiters of the space of an informative artifact.

No	Problem	Solution	Name	N°
01	How much of the content of an informative artifact should reflect in a window?	Use the <i>Task Window Pattern</i> . Use correspondent patterns of this proto-pattern language instead of related patterns of the <i>Task Window Pattern</i> .	Window (Shape) Pattern	14
02	How do you represent the content of a window of an informative artifact, from several points of view?	Use the <i>Dividing the Window into a Few Panes Pattern</i> . Use pictorial patterns of the visual arts to compose your panels.	Panel (Surface) Pattern	15

* Number of a pattern in the set of patterns of my proto-pattern language

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 3: Pattern 1:

Window (Shape) Pattern

Name**Window (Shape) Pattern****Problem**

How much of the content of an informative artifact should reflect in a window?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the *Spatial Pattern* to develop a spatial strategy for defining the properties of the space of your artifact.

You applied the *Organizational Patterns* to organize the space of your artifact. You want to visualize your artifact through windows and you know that there is a pattern that can be applied (the *Task Window Pattern* [Kent, 95])⁸⁰

Forces

Any solution to this problem should balance the following forces:

Conflict between Definitions: The definition of the shape in the spatial model of your informative artifact, and the concept of a window in software design may have conflict.

Conflict between Sequential Arrangement and Windowing: If you use a sequential arrangement to organize content of your informative artifact, this arrangement may have some conflict with the concept of windowing.

Unconformity of the Related Patterns: Related patterns of the *Task Window Pattern* may not fit into the organization of this proto-pattern language.

Solution

Define a window as “*an abstract representation of the content, or part of the content of the informative artifact.*” Use the *Task Window Pattern*.

This approach may result in unconformity of the related patterns of the *Task Window Pattern* with the organization of this proto-pattern language.

To solve this problem you may replace the *Finding a Task Pattern* with the *User Participation Pattern*, the *Available Actions Pattern* with *Generic Forms Patterns*, and the *Dividing the Window into a Few Panes* with the *Panel (Surface) Pattern*.

Rationale

The definition of the shape in the spatial model of your informative artifact is similar to the concept of a window in software design. In the pattern based architecture shape is an abstract representation of the content. ⁹⁰ Software design uses the similar concept to define a window. ⁹¹

The sequential arrangement of the content can be layered by task oriented windowing arrangement. It means that each sequence can be displayed in multiple task windows. Also, a task window can display multiple sequences. This is a layering technique.

Known Uses

Implementing Technology Transition. [Dart, 94]

Related Patterns

Task Window Pattern [Kent, 95],

Generic Forms Patterns (of this proto-pattern language),

Panels (Surface) Pattern (of this proto-pattern language.)

User Participation Pattern (of this proto-pattern language)

Sketch

Not applicable.

Example

Suppose that you want to design an educational CD that contains many

different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.⁹² You applied ***Organizational Patterns*** to organize the space of your artifact. You want to visualize your artifact through windows.

Your CD has three parts (Introductory, descriptive, and testing). If in the descriptive parts a user should refer to a document on the web, by applying the ***Task Window Pattern*** you will have two windows in your CD. One is the window of CD, and the other is a navigator.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 3: Pattern 2:

Panel (Surface) Pattern

Name

Panel (Surface) Pattern

Problem

How do you represent the content of a window of an informative artifact, from several points of view?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied ***Organizational Patterns*** to organize the space of your artifact. You applied the ***Window (Shape) Pattern*** to visualize the content of your artifact through windows. You know that you need to have a substitute for its related pattern "the ***Dividing the window into a Few Panes*** [Kent, 95] " to represent the content of a window from several points of view. ⁹³

Forces

Any solution for to the problem should balance the following forces:

Conflict between Definitions: The definition of the surface in the spatial model of your informative artifact, and the concept of a panel in software design may have conflict.

Language dependency of the “Dividing the Window into a Few Panes” Pattern: This pattern is written for Smalltalk and pane is a concept that belongs to that language. Therefore, this pattern may not be applicable in other languages or at the interaction domain.

Esthetic Considerations: You can display the panels of a window simultaneously or continuously. The panels in both situations may have a good or bad composition.

Solution

Define a panel as *“an abstract representation of the content, or part of the content of the informative artifact.”* There is no difference between the concept of a pane and a panel in software design. Use the ***Dividing the Window into a Few Panes Patterns.***

Use compositional theory and experience of the visual arts (pictorial patterns)

to compose your panels. ⁹⁴

Rationale

The *surface* in the spatial model of your informative artifact (the pattern based architecture) is a delimiter of the shape and first visual representation of the space. ⁹⁵ Software design uses the similar concept to define the concept of a panel. ⁹⁶

The concept of the pane in Smalltalk and panel in other programming languages like Java is the same. Therefore you can use the *Dividing the Window into a Few Panes Patterns* in both cases and in this proto-pattern language.

In the pattern based architecture, the surface is the subject of pictorial patterns. Therefore, the esthetic criteria should be applied to the surface.

Known Uses

Implementing technology transition [Dart, 94]

Related Patterns

Dividing the Window into a Few Panes Pattern, [Kent, 95]

Window (Shape) Pattern (of this proto-pattern language.)

Sketch

Not applicable.

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the

spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact. ⁸⁷

You applied ***Organizational Patterns*** to organize the space of your artifact. You applied the ***Window (Shape) Pattern*** to visualize the content of your artifact through windows and you know that Your CD has a display window and a browser window. ⁸⁸

Your CD may have three parts (Introductory, descriptive, and testing). If you want to show a multiple view of the content of the CD in its testing part (for example four tests together) in the display window of your CD, you can apply the ***Panel (Surface) Pattern*** and divide the window into four panels. Each panel shows a test. To have a good composition of these panels you can use an artistic composition.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 4:

Generic Forms Patterns

This set of patterns refers to the generic forms that can create a representation of the space of an informative artifact.

No	Problem	Solution	Name	N*
01	How do you create a visual representation of your informative artifact using conceptual widgets?	Use " <i>A Pattern Language for Human-Computer Interface Design</i> " and " <i>Iconic Reference Pattern</i> ."	Conceptual Widgets Pattern	16
02	How do you create a visual representation of your informative artifact and conceptual widgets?	Read the specification of the technical (programming) language of your artifact, to have a general idea of its abilities and restrictions.	Concrete Widgets Pattern	17

* Number of a pattern in my proto-pattern language

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 4: Pattern 1:**

Conceptual Widgets Pattern

Name

Conceptual Widgets Pattern

Problem

How do you create a visual representation of your artifact using conceptual widgets?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact. You applied ***Organizational Patterns*** to organize the space of your artifact. You applied ***Delimiters Patterns*** to delimit the space of your artifact and represent it through some windows and its panels. To represent the content of the artifact to the user, you need to use some widgets. You know that there are some patterns for implementing widgets.

Forces

Any solution to the problem should balance the following forces:

Different kinds of Widgets: A widget may be conceptual or concrete, just like a word in a natural language, such as “Wisdom” and “Door.” This is the point that interaction design (or interaction model) and software design (or application model) meets each other.⁹⁹ In other words, it is the conjunction of social and technical patterns. Therefore, the same kinds of patterns or pattern languages cannot apply to both. Widgets have a hybrid nature.

Insufficiency of an Existing Pattern Language: You may choose an existing pattern language to apply in your artifact for creating its widgets. But any pattern language may be insufficient and does not solve all of the problems that you may be faced in developing your artifact.

Choosing an Existing Pattern Language: The criterion for choosing a pattern language is the number of problems that the application of the language can solve. But there may be some more sophisticated solutions that are not included in the language of your choice. Also, there may be some patterns for solving some problems that your selected pattern language does not cover.

Solution

Separate the conceptual and concrete widgets from each other. Use “**A Pattern Language for Human-Computer Interface Design**” [Tidwell, 98] and

Iconic Reference Pattern [Azad, 98] to implement the conceptual widgets.

If you faced a problem that its solution is not presented in the **A Pattern Language for Human-Computer Interface Design**,¹⁰⁰ use only those related patterns that are accepted by the professional bodies of the pattern community.

Rationale

Organizational Patterns are the same as the grammar of a language. The **Delimiters Patterns** are paragraphs and sentences. To make a sentence you need words, or widgets in the case of an informative artifact. Any word may refer to a subjective or an objective concept. Therefore, there are some conceptual (such as an icon) and some concrete (such as a button) widgets.

There is a pattern language that refers to the conceptual widgets.¹⁰¹ You do not have any reason to write another one. If you have any comment about that you can send those to the authors. If you cannot find a solution to the problem you are faced in that pattern language, use the patterns that are supported by the professional bodies of the pattern community. You can find many patterns on the Web, but you cannot guarantee that they are the best solution to your problem.

Known Uses

As it is explained in the documentation of each pattern of **"A Pattern Language for Human-Computer Interface Design"** and **Iconic Reference Pattern**.

Related Patterns

Concrete widgets Pattern (of this proto-pattern language)

Sketch

Not applicable

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the **Spatial Pattern** to develop a spatial strategy for defining the properties of the space of your artifact.¹⁰² You applied **Organizational Patterns** to organize the space of your CD.¹⁰³ You applied **Delimiter Patterns** to delimit the space of your CD and represent its content through some windows and its panels.¹⁰⁴ To represent the content of the CD to the user, you need to use some widgets including a text entry in the main node of the testing part of your CD. You are facing the problem of how your CD should indicate the kind of information that should be supplied. You can use the **Forgiving Text Entry Pattern** (pattern 40 of *A Pattern Language for Human-Computer Interface Design*) and allow the user to enter the text in any recognizable format for your CD.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 4: Pattern 2:

Concrete Widgets Pattern

Name

Concrete Widgets Pattern

Problem

How do you create a visual representation of your artifact and conceptual widgets?

Context

You want to design a navigable and interactive informative artifact. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.

You applied ***Organizational Patterns*** to organize the space of your artifact. You applied ***Delimiters Patterns*** to delimit the space of your artifact and represent it through some windows and its panels. To represent the content of the artifact to the user, you applied the ***Conceptual Widget Pattern***. At this point you need to create a concrete feature for your conceptual widgets.

Forces

Any solution to the problem should balance the following forces:

Application's language dependency: Any application has some language dependency that imposes some restriction on your artifact. Also, any language has some special abilities.

Common Literature: Although interaction design and application design have much common ground, they do not speak in the same language.

You need to have some basic information about application design to communicate with an application designer, who designs the infrastructure of your artifact.

Solution

Read the specifications of the application's language to have a general understanding of the restrictions and the abilities of that. Participate in technical discussions and events related to the application design.

Rationale

You do not need to be expert in the language or design of an application, but you need to have a general understanding of them. This understanding will help you to use the abilities of the language or design of application and face its restrictions. Also, you will be able to talk effectively to application designers.

Known Uses

Social Technical Management Approach,

The exchange of ideas between Architects and Structural Engineers.

Related Patterns

None.

Sketch

Not applicable

Example

Suppose that you want to design an educational CD that contains many different kinds of information. You wish to use the architectural space as the spatial model of your artifact. You applied the ***Spatial Pattern*** to develop a spatial strategy for defining the properties of the space of your artifact.¹⁰⁶ You applied ***Organizational Patterns*** to organize the space of your CD.¹⁰⁶ You applied ***Delimiter Patterns*** to delimit the space of your CD and represent it through some windows and its panels.¹⁰⁷ You applied the ***Conceptual Widgets Pattern*** to represent the content of the CD to the user.¹⁰⁸ If you want to use Java for creating your CD, you need to have some general information about Java, otherwise how can you talk to the programmer that should write the codes for your CD?

The most conventional way is giving the prototype of your design to her to be implemented, and go through a lot of undesired changes.

A Proto Pattern Language for Human-Computer Interaction

Chapter 4: Section 3: Part 4:

Social Patterns

This pattern refers to the application of user participation principle in the space of an informative artifact.

N	Problem	Solution	Name	N*
01	How do you facilitate the participation and intervention of the user in an informative artifact?	Work with the user. Create a variety of solutions. Use a minimum functionality approach. Implement your findings through the process of creating your artifact.	User Participation Pattern	18

* Number of a pattern in the set of patterns of my proto-pattern language

*A Proto Pattern Language for Human-Computer Interaction***Chapter 4: Section 3: Part 4: Pattern 1:**

User Participation Pattern

Introduction

As computer systems became more complex business emphasized on the quality than the productivity. The traditional water flow life cycle methodologies could not satisfy these trends. Joint Application Development (JAD or User Centric Design) and Participatory Design (PD) methods were proposed in order to address the problem. These two methods both emphasize greater user involvement and user participation in the development of systems. Joint Application Development was originally adopted in North America, while Participatory Design in Scandinavia. Joint Application Development is a structured approach. It has five well-defined phases which every Joint Application Development session should include. The Joint Application Development session is facilitated by the facilitator who controls the pace of the session. There is an agenda for each session which set the goals and issues that the session should accomplish. The focus of the Joint Application Development is more on economic and performance.

Participatory Design recognizes that design is fundamentally a social activity, is not entirely rational, and that it is a process of reflection-in-acting best done in an iterative manner. The focus of Participatory Design is to improve workplaces. It emphasizes more the social effect of the system. To claim that there is no "standard" or fixed order of practices in Participatory Design that one can follow, is a representation of having a static and reactive attitude(in contrast to dynamic and proactive approach of Participatory Design) toward the system design.

Alexander's pattern theory, and pattern movement (in architecture or software) follow a Participatory design principle. They look at the system as a **Social-Technical system**. Joint Application Design defines a system as a **Technical-Social system**. Therefore, in spite of its claim to focus on "User Centric Design," Joint Application Design approach emphasizes on the performance more than the user needs. ¹⁰⁰

Participatory Design, Alexander's approach, and Pattern movement (in architecture and software) support a more sociological view of design. The core idea of these three movements can be summarized in the following sentence *"Design should be done with users, neither for nor by them."* Joint Application Development (User Centric Design) which advocates a physiological view claims that *"Design should be done for users, neither by nor with them."*

Name

User Participation Pattern

Problem

How do you facilitate the participation and intervention of the user in an informative artifact?

Context

You want to design a navigable and interactive informative artifact. You want to use the architectural space as the spatial model of your artifact. You applied the **Spatial Pattern** to develop a spatial strategy for defining the properties of the space of your artifact. You know that the spatial model of your artifact strictly follows the user participation principle. The user participation principle has two main features including participation through the process of creating an artifact, and during the everyday use of the artifact. You can use a Joint Application Development methodology or a Participatory Design methodology to facilitate the participation of a user through the process of creating and using an informative artifact.

Forces

Any solution to the problem should balance the following forces:

Weaknesses of Joint Application Development Methodology: The

main weaknesses of Joint Application Development methodology are the restrictions that it imposes on the user. In Joint Application Development the end users are ignored as participants in meetings because of an unjustified lack of confidence that the end user can meaningfully contribute to the design process. Consequently, this results in a meeting room filled with middle managers, technical managers, and front-line users who are unable to specify details of day to day use.

Although Joint Application Development practitioners utilize many creative techniques and paraphernalia in the design process, it was found that often these are not sufficient to guarantee the creative participation of a user in developing and using an artifact.

Weaknesses of Participatory Design Methodology: It is supposed that Joint Application Development method emphasizes structure, while the Participatory Design approach devotes almost no guidelines to structure. A rigid structured method does not facilitate creativity. The people who advocate the combination of Joint Application Development method and Participatory Design claim that using a structure which preserves the contextual flexibility can actually can reflect the creativity of the end user. Moreover, they believe that preparing a guideline for using Participatory Design method will push this method further into the hands of the average

designers in the industry.

Restrictions of maximum functionality approach: A maximum functionality approach creates a sophisticated version of the artifact. The artifact seems so perfect that a user is afraid to change it. She will try to learn to work with the artifact. Therefore, it seems to her that the artifact tends to impose its own rationale and taste and rationale on her.

Ignoring findings through process: By using a Joint Application Development methodology, the final document is a blue print that should be kept untouched. If a designer through the process finds some new way or ideas that are generated by the gained experience, she should ignore them or interpret them in the same direction as it is documented in the final document.

Solution

To facilitate participation of the user in the design of your artifact, use a Participatory Design approach. To do so follow these steps:

1- Work with the user from the beginning of the process. Give her your pen to draw her ideas of the artifact for you. Make her a part of your production team.

2- Visualize her ideas in the best way.

3- Visualize the possible options for the user.

4- Create prototypes of a variety of solutions to present to the user and let her evaluate the concrete options.

To facilitate participation of the user in your artifact during its use, follow these steps:

1- Create an artifact with minimum functionality.

2- Create a list of all possible options for user participation through your artifact.

3- Implement necessary tools to facilitate her intervention.

Recursively, use the all previous steps to implement your findings through the process of creating your artifact. Do not undermine or interpret them in away that support your first draft.

Rationale

In Participatory Design you work with the user and customer from the start to the end of the process of developing an artifact. Therefore, you do not need to go through expensive and time consuming usability testing procedures. By applying minimum functionality approach you let the taste and rationale of a user be predominant on your artifact. An informative artifact is not an artistic product, it should look like what a user wants, and respond to all her needs.

Known Uses

WindowPerTask Pattern [Back, 87], ¹¹⁰

Using Expert Domain technique for developing a software application.

[Fowler, 96],

Writers Workshop Patterns. [Coplien, 98] ¹¹¹

Related Patterns

All patterns of this proto-pattern language.

Sketch

Not applicable.

Example

Suppose that you want to design an educational CD that contains a lot of different kinds of information. You wish to use the architectural space as the spatial model of your CD. You applied the **Spatial Pattern** to develop a spatial strategy for defining the properties of the space of your CD. ¹¹²

Make the customer (the teacher) and a potential user (a student) member of your team. Let them design it for you. Apply **Organizational patterns**, **Delimiter patterns** and **Generic forms Patterns** to implement your CD considering their ideas.

Prepare prototypes of different versions of your CD. Compare those and let them listen to your rationale. Let them to criticize and correct you. In spite of any

scientific or artistic value that your own rationale and taste can have, if your user cannot understand, enjoy or like your product, your CD is not an informative artifact.

During the implementation prepare a list of all possible options of user participation in your CD. Use ***Generic Forms Patterns*** to implement necessary tools for activation of these options. For example, if a user wants to change the color of background of the introductory part, she can drop down a menu and choose her favorite color.

If through the process you or the users that are working with your team find out that your CD should have four parts, add that part (for example a feedback part.) If it is your finding try to explain it to the users and convince them that your CD should have another part.

*A Proto Pattern Language for Human-Computer Interaction***Chapter 5:**

Conclusion

Achievements
Future Works

*A Proto Pattern Language for Human-Computer Interaction***Chapter 5:**

Conclusion

My thesis set out to develop a proto-pattern language for human-computer interaction that uses the pattern based architectural space as its abstract spatial model and takes into consideration both technical and social aspects of an informative artifact.

In chapter one I explained the structure of my thesis, and in this chapter I compare the results of my work with my original goal to show how the goal was achieved, what new subjects have emerged from the process of developing this proto-pattern language, and how the work can be continued in the future.

Achievements:

My thesis coined the term of "Informative Artifact" as an artifact that is giving information, and expanded its meaning to cover all artifacts that are developed by the help of computer applications and facilitate any interaction between the user and a computer. I claimed that all of these artifacts and interactions are informative.

The goal of my thesis had a three-part structure:

- 1- Using the pattern based architectural space as the abstract spatial

model of an informative artifact,

2- Taking into consideration technical aspects of an informative artifact,

3- Taking into consideration social aspects of an informative artifact.

To fulfill the first part of my goal, I developed a proto-pattern language, which includes 18 patterns that are connected to each other by an organizational path. My proto-pattern language begins with space modeling patterns followed by Organizational Patterns that can organize the space of an informative artifact. All of these patterns work at the abstract level. Spatial Patterns define the strategy, and Organizational Patterns illustrate the policies for organizing the space of an informative artifact. These abstract patterns are realized into a concrete feature by the help of Delimiters and Generic Forms Patterns. To realize the abstract patterns into concrete patterns, I followed the same procedure (theory of representation) that the pattern based architecture follows to delimit and realize its abstract values.

To guarantee that all technical considerations for developing an informative artifact are regarded and fulfill the second part of my goal, I used the vocabulary and conventions of the software pattern community, as well as the related patterns and pattern languages that are written by this community. I adopted the existing patterns and did not try to rewrite them. I could not find a good reason to change them; they work very well as experiences of the pattern community show.

These patterns and pattern languages include the WindowPerTask and Dividing Window into a Few Panes Patterns, [Kent, 95] "A Pattern Language for Human Computer Interaction Design," [Tidwell, 98] and the Iconic Reference Pattern. [Azad, 98]

According to my definition of the transparency principle an informative artifact should present a clear view of the functionality of its background system, and the user's location in the artifact. I found that following the transparency principle strengthens the ability of an informative artifact to reflect the inner principles of the related technology (background system.)

Since the existing version of the State Pattern [Gamma, et al., 95] refers to the inner logic of the background system of an informative artifact, by following transparency principle, I concluded that I should interpret the State Pattern to have a representation at the interaction domain and be applicable to an informative artifact.

To guarantee that all social considerations are regarded and fulfill the third part of my goal, my proto-pattern language through its User Participation Pattern advocates the application of Participatory Design instead of the use of Joint Application Design approach.

When I began to write my proto-pattern language, I knew that I should facilitate the user participation, but I was not aware that the current challenges in

the field human-computer interaction are mostly the result of application of Joint Application Design approach. Therefore, returning to Participatory Design in this proto-pattern language should be considered a result of the process of writing the language and its patterns.

What is Next?

There are some ideas in my thesis that I referred to briefly, but I have not documented the details. I did it intentionally to focus on the most related topics to the goal of my thesis and to keep its interdisciplinary nature on track. Any of these ideas may be the target of a further research and/or documentation. I categorized those ideas based on the main discipline to which they are related as follows:

In Architecture:

Up to 1995 I have lived in a pattern based architectural space. My twenty years of direct experience in working with patterns as an architect, has proved to me that Alexander was correct in asserting that *beauty is objective* and is not in the eyes of the beholder. As the readers of the drafts of his unpublished book noted, ¹¹³ he now claims that beauty is geometric.

I do not agree with Alexander upon geometrical nature of beauty. I have learned from the pattern-based architecture that *beauty is the solution*. A

solution is a product of human intellect, when we encounter its realization we admire that shared humanistic value of intellect regardless of the level of our familiarity with the subject or context. For this reason beauty is universal.

According to my experience (that I referred to above) I do not agree with the Alexandrian definition of "quality without the name." This quality can be defined as "the representation of the knowledge of simultaneous presence of the upper bound and the lower bound restrictions of your universe at any slot of the time." In this definition quality without a name is the quality of the Time/Space.

I do not agree with the strong tendency in architecture that advocates any direct relationship between philosophy and architecture. According to my experience with the pattern based architecture, as Alexander indirectly ¹¹⁴ concluded, the path of exchange of ideas between these two disciplines passes through the public opinion which is a value system (ritual). Ritual has some similarities to Participatory Design, but is in contrast with Joint Application Design approach.

Implementing and documenting these three topics (Esthetic Theory, Qualitative Values, and Design Approach of the pattern based architecture) are important to support the theoretical background of this

proto-pattern language, and should be done in the future.

The summary of my findings through direct experience with the pattern based architecture are included in **Appendix 1: *Patterns in Architecture*** under the heading of: "Fundamental Concepts of the Pattern Based Architecture."

In Human-Computer Interaction:

I have doubt that human-computer interaction is a multi-disciplinary field.

The fact that many people from different backgrounds are working in this field does not necessarily means that it is a multi-disciplinary task.

No architectural product can come to the world without using the expertise of different disciplines such as Civil and Electrical Engineering. Therefore, architecture has a multi-disciplinary nature. This is not true about human-computer interaction. Any interface for an informative artifact can be designed and implemented by a software designer if he considers the rules of information organization in his product and applies the principles of participatory design. It means that creating an interface is not an act of visual art. It also means that the task of "research and creating a stereotype of the user as a human person" can be omitted by putting the pen in the hand of the (expert domain and non-professional) end users and let them to participate in the design of an informative artifact. I should

emphasize that I do not mean that design should be done by the user but that I strongly advocate the concept of design with the user.

I think that the application of participatory design approach has mutual benefits for the industry and end users. This approach decreases the cost and development time of the product and increases the chance of the users getting what they really want. To apply the principle of participatory design in the field of human-computer interaction, we need to shift from a physiological attitude toward the end user to a social attitude and bridge the great divide. ¹¹⁶

Therefore, research and evaluating the theoretical background of the participatory design approach, its existing techniques and procedures, and the study of the result of its previous applications, are essential to support the basic claim of my thesis to advocate the application of participatory design in the field of human-computer interaction and should be done in the future.

The summary of my findings about the theoretical background, and existing techniques and procedures of participatory design are included in **Section 7 of Appendix 5: “Joint Application Development and Participatory Design.”**

Technical Aspects:

As I mentioned In the Scalability Pattern of the proposed proto-pattern language, It is necessary to find the threshold on which you should begin to measure the complexity of an informative artifact numerically. Also, it is necessary to develop a simple technique for this measurement.

Tidwell's work "A Pattern Language for Human Computer Interaction Design" [Tidwell, 98] as it is acknowledged by its writer, needs some revision and should be completed. The Iconic Reference Pattern [Azad, 98] also needs some revision. The correction of this pattern depends on the correction of the previous one.

Therefore, from the technical point of view, finding the threshold on which we should begin to measure the complexity of an informative artifact numerically, developing a simple technique for this measurement, revision and completion the "A Pattern Language for Human Computer Interaction Design" and "Iconic Reference Pattern," are necessary to support the technical aspects of my proto-pattern language and should be done in the future.

The summery of my findings about the theoretical background of the complexity theory are included in **Section 1 of Appendix 5:**

"Organization and Complexity."

Also, a summery and a critical review of Tidwell's pattern language and a brief description of the Iconic Reference Pattern are included in Section 6 of Appendix 5: *"Widgets and Patterns."*

Furthermore, my proposed proto-pattern language should be tested by experience. It should be applied in the real world to be evaluated and gone through further corrections.

A Proto Pattern Language for Human-Computer Interaction

Appendices

Appendix 1: Patterns in Architecture

Appendix 2: Christopher Alexander's Pattern Theory

Appendix 3: Patterns in Software

Appendix 4: Patterns in Human-Computer Interaction

Appendix 5: Related Patterns

*A Proto Pattern Language for Human-Computer Interaction***Appendix 1:****Patterns in Architecture**

Design paradigm
Fundamental Concepts of "Pattern Based Architecture"

A Proto Pattern Language for Human-Computer Interaction

Appendix 1:

Patterns in Architecture

To clarify what I mean by the **"Pattern based architecture"** I will briefly look at its basic ideas and historical background.

Before the advent of the **Renaissance** in the *west* and the spread of the **Modern mentality** in the *east* from the end of last century, artifacts tended not to suffer from adaptation, quality, and usability failures.¹¹⁶ The "unselfconsciously" constructed artifacts of tradition were produced without the benefit of formal models and methods. A system of implicit and often inflexible rules for design and construction progressed in an evolutionary fashion. Over time, natural forces caused successive artifacts to better adapt to and mesh with their environments, almost always finding points of equilibrium and beauty. This resulted in increasingly better rules applied by people who did not necessarily know why the rules worked.

Design Paradigm

Historically, the modern "rational" design paradigm was both a contributing factor towards and a byproduct of the professionalization of design.¹¹⁷ Rational

design is distinguished from traditional craftsmanship by its "self-conscious" separation of designs from products, its use of analytic models, and its focus on methods that anyone with sufficient formal training may apply. Analytic designers first make tractable models (from simple blueprints on up) that are analyzed and manipulated into a form that specifies construction.

Rational design was in many ways a major advance over traditional methods. However, the notions of analysis and synthesis are construed in architecture and artifact design, leading to the sterile study of methods that have no bearing on the vast majority of artifacts actually built or the work involved in developing them. Wolfe in "From Bauhaus to Our House" refers to this fact. [Wolfe, 81]

The main problem lies in separating activities surrounding analysis and synthesis rather than recognizing their duality. While it is common to exploit the symmetries between *form* and *function* ¹¹⁸ (roughly translatable as system statics versus dynamics) further opportunities for integrating them become lost. Like an organism, a building is more than a realization of a design or a development process. Model, process, context, and artifact are all intertwined aspects of the same system. Artificial separations of models, phases, and roles break these intertwined connections.

Fundamental Concepts of "Pattern Based Architecture"

The pattern based architecture creates a spatial organization where the

space is qualified by the forms that exist. This space is not a Cartesian or Euclidean space. This organized space is structured, continuous, hierarchical, and has orientation (with an abstract reference point) and an abstract view. Time has two properties: Motion and Rhythm. Time and Form are simultaneous continuities. In this Time-Space beauty is objective and depends as much on geometrical order as on the ability to reflect a higher and more profound order. Beauty is depending on the conformity of all aspects of a form with the proportions inherent in the nature.

In the pattern based architecture *shapes* are delimiters of the space. Shape is numeric, where numbers are qualitative as well as quantitative entities and geometry is the expression of their personality. Everything has a form that is observable through mathematical laws of similitude, symmetry, and geometry. Order and proportion are cosmic laws comprehended through arithmetic, geometry, and harmony. Mathematics of proportion follows the proportional series of golden mean. The units of spatial definition are members of the human body. The *surface* is a delimiter of shape that has a twofold function. The surface contains "sense of place" and is continuous. Continuity at this level is achieved by applying pictorial patterns. Pictorial patterns of the surface may be single layer or multi layers, and each may follow a Geometric, an Arabesque, or a Calligraphy order. Delimiters and their subclasses have temporal nature.

In this space colors that can create order where otherwise chaos might exist have seven, three, four colors pallet systems. Adjusted or opposite colors may be used in a single layer or in multi-layers. Matter as the passive complement of intellect has four symbolic properties and real aspects.

In the pattern based architecture space is a synthesis of generic forms with some natural and symbolic elaboration. To synthesize the generic forms, a natural, geometric or harmonic order or any sub-order of those may be used in correspondence to the scale of the space. This space is the subject of a mutual process of growth and development, depending on its capacity. In the pattern based architecture changing an order depends on an external motivation and internal capabilities.

The pattern based architecture advocates a public and on-the-fly decision making system and cannot be learned in the school. It reflects a value system, but it is independent of that value system. They are tidily woven to each other and other social systems by the means of ritual and tradition. ¹¹⁹

Two examples of pattern based architectural space that represents its inner harmony and organization is shown in Figure 31. ¹²⁰

Kashan (Bazar)

Esfahan (Bazar)

Figure 31: Two examples of pattern based architectural space.

*A Proto Pattern Language for Human-Computer Interaction***Appendix 2:**

Christopher Alexander's Pattern Theory

Section 1: Christopher Alexander's Biography and Publications

Section 2: Alexander and evolution of an idea

Section 3: Evaluation of Alexander's ideas

*A Proto Pattern Language for Human-Computer Interaction***Appendix 2: Section 1:****Christopher Alexander's****Biography and Publications**

**Biography
Publications**

A Proto Pattern Language for Human-Computer Interaction

Appendix 2: Section 1:

Christopher Alexander's

Biography and Publications

Biography

© 1994 max kisman / maxk@berkeley.ca

Figure 32: Alexander by Max Kisman

Christopher Alexander was born in Vienna, Austria, and raised in Oxford and Chichester, England.¹²¹ He graduated from Cambridge University, where he studied Mathematics and Architecture. He then obtained a Ph. D. in Architecture at Harvard University. For his PhD thesis, later published as the book *Notes on the Synthesis of Form*, he was awarded the first Gold

Medal for Research by the American Institute of Architects.

Since 1963 Alexander he has been Professor of Architecture at the University of California at Berkeley, and Director of the Institute for Environmental

Structure. In 1980, Alexander was elected as a member of the Swedish Royal Academy. In 1996 he was elected as a fellow of the American Academy of Arts and Sciences. Christopher Alexander is a Trustee of the Prince of Wales's Institute of Architecture.

Christopher Alexander is the author of among others *The Timeless Way of Building* and *A Pattern Language: Towns, Buildings, Construction*. He and his colleagues at the Center for Environmental Structure have published a major statement in the form of these books that, in their words, "lays the basis for an entirely new approach to architecture, building and planning, which will replace existing ideas and practices entirely." ¹²²

Publications:

The following list includes most of the Alexander's publications: ¹²³

* **The Use of Diagrams in Highway Route Location: An Experiment** ¹²⁴

Christopher Alexander and Marvin L. Manheim

Cambridge, School of Engineering, Massachusetts Institute of Technology, 1962

* **HIDECS 1: A computer program for the hierarchical decomposition of a set that has an associated linear graph**

Christopher Alexander and Marvin L. Manheim

Cambridge, School of Engineering, Massachusetts Institute of Technology, 1962

* **HIDECS 2: A computer program for the hierarchical decomposition of a set that has an associated linear graph**

Christopher Alexander and Marvin L. Manheim

Cambridge, School of Engineering, Massachusetts Institute of Technology, 1962

* **HIDECS 3:** Four computer programs for the hierarchical decomposition of systems that have an associated linear graph.

Christopher Alexander

Cambridge, Mass. MIT, 1963

Notes on the Synthesis of Form**Christopher Alexander**

Cambridge, Harvard University Press, 1964

Community and Privacy; Toward a New Architecture of Humanism**Serge Chermayeff, Christopher Alexander**

Harmondsworth, Penguin, 1966

*** The Atoms of Environmental****Christopher Alexander, Barry Poyner**

Berkeley, Center for Planning and Development Research, University of California, 1966

*** The City as a Mechanism for Sustaining Human Contact****Christopher Alexander**

Berkeley, Center for Planning and Development Research, University of California, 1966

A Pattern Language, which Generates Multi-Service Centers**Christopher Alexander, Sara Ishikawa, Murray Silverstein**

Berkeley, Calif., Center for Environmental Structure, 1968

Houses Generated by Patterns**Christopher Alexander**

Berkeley, Calif., Center for Environmental Structure, 1969-1 v.

The Oregon Experiment

Christopher Alexander, et al.

New York: Oxford University Press, 1975

A Pattern Language: Towns, Buildings, Construction

Christopher Alexander, Sara Ishikawa, Murray Silverstein, with Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel

New York: Oxford University Press, 1977

The Timeless Way of Building

Christopher Alexander

New York: Oxford University Press, 1979

The Linz cafe = Das Linz Cafe

Christopher Alexander

New York: Oxford University Press; Wine: Locker Verlag, 1981

The Production of Houses

Christopher Alexander, Howard Davis, Julio Martinez, Donald Corner

New York: Oxford University Press, 1985

A New theory of urban design

Christopher Alexander, et al.

New York: Oxford University Press, 1987

A Foreshadowing of 21st Century Art: the Color and Geometry of very early Turkish Carpets

Christopher Alexander

New York: Oxford University Press, 1993

Domestic Architecture ¹²⁵

Doors of Perception 2 @HOME Conference

4-6 November 1994, Amsterdam

<http://www.mediamatic.nl/Doors/Doors2/Alexander/Alexander-Doors2-E1.html>

The Mary Rose Museum

Christopher Alexander, Gary Black, Miyoko Tsutsui

New York: Oxford University Press, 1995

The Nature of Order

Christopher Alexander

New York: Oxford University Press, 200?

*A Proto Pattern Language for Human-Computer Interaction***Appendix 2: Section2****Christopher Alexander and****Evolution of an Idea**

Notes On The Synthesis of Form**The Oregon Experiment****A Pattern language: Towns, Buildings, Construction****The Timeless Way of Building****A New theory of Urban Design: A Foreshadowing of 21st Century Art: the color
and geometry of very early Turkish carpets** **The language of the carpets****Domestic Architecture****The Nature of Order**

A Proto Pattern Language for Human-Computer Interaction

Appendix 2: Section 2:

Christopher Alexander and

Evolution of an Idea

In this section I will have a brief look at eight important books, essays and projects that are published or designed by Alexander during the last 35 years. It may be useful to find out how his idea of a modern pattern Language came to life. The list of publications that will be reviewed includes: *Notes on the Synthesis of Form*; *The Oregon Experiment*; *A Pattern Language: Towns, Buildings, Construction*; *The Timeless Way of Building*; *A New Theory of Urban Design*; *A Foreshadowing of 21st Century Art*; *Domestic Architecture*; *The Nature of Order* his last book that is under publication.

Notes on the Synthesis of Form

In this book, Alexander shows that in the design process, a design stabilizes when there are no more "misfit variables." ¹²⁸ The only (practical) way that this will happen is for the design to stabilize within each strongly connected component before stabilizing among more weakly connected sets of components.

In his epilogue, he states: "My main task has been to show that there is a deep and important underlying structural correspondence between the pattern of

a problem and the process of designing a physical form which answers that problem."

Alexander strove to use software for semi-automated urban design in his early work. In this book he built tools to lower the cognitive load of design by exploring large design spaces on behalf of the architect. Patterns were in part a reaction to the shortcomings of this approach.

In Appendix I, he shows a worked example from "The determination of Components for an Indian Village." An agricultural village of six hundred people is to be reorganized to make it fit present and future conditions developing in rural India. (The full paper appears in Conference on Design Method, Oxford: Pergamon Press, 1963, Christopher Jones, Ed.) He lists all the needs, assertions, and rules as potentially misfit variables. Then he tabulates all the interactions between variables. He then analyzes in detail the decomposition of the resulting graph, showing the tree structure of the village and an account of each diagram and associated village functions.

Appendix II gives the mathematical treatment of his design algorithm. In 1963 Alexander published reports describing "The hierarchical decomposition of systems which have an associated linear graph." These four HIDECS or computer programs were written in IBM 7090 assembly code! It would seem that these algorithms merit being incorporated into design tools.

The Oregon Experiment

This book is the first of the series to describe in full detail how Alexander's theory may be implemented.¹²⁷ It explains a new planning process for the University of Oregon where an entire community of 15,000 persons is, through the means described in this book, now taking charge of all future planning and design.

At the core of these series is the idea that people should design for themselves their own houses, streets, and communities. This idea may be radical and implies a radical transformation of the architectural profession but it comes simply from the observation that most of the wonderful places of the world were not made by architects but by the people.

This book is the documentation of the ideas and process of design of a master plan for the University of Oregon and is implemented at that university, but it shows at the same time how any small community as a university or a small town might go about designing its own future environment—with all the members of the community participating personally. It is a concrete example of the Center's theories in practice, showing in simple detail with numerous illustrations, how to implement six guiding principles: organic order, participation, piecemeal growth, patterns, diagnosis, and coordination.

Presently, The University Planning Office guides and manages the planning

processes associated with the University's physical development. As part of this function, this office staffs the Campus Planning Committee, maintains an inventory of space use, and manages capital improvement projects during design. The presence and activities of this office are inspired by Alexander's ideas. Also, the current planning process as documented by that office follows the Alexander's proposed principles.

The University of Oregon's current planning process (as articulated in the Long Range Campus Development Plan of 1991) is based on principles adopted by the University in 1974. The principles have come to be known as "The Oregon Experiment" and are also the subject of a book of the same name by Christopher Alexander of The Center for Environmental Structure in Berkeley California.

The history of development of the University of Oregon campus in Eugene began with Ellis Lawrence's preparation of a "Block Plan" of the campus in 1914. Lawrence revised this initial effort in 1923 and prepared a major modification in 1932.

The concepts of spatial organization contained in these early plans, which were reflective of Lawrence's Beaux-Arts training, are still evident on this campus seventy-five years later. Moreover, the policies expressed in this current document are intended to preserve and expand the network of interconnected quadrangles, squares, malls, and promenades which were characteristic of

Lawrence's early development pattern.

In 1962 the University selected urban designer Lawrence Lackey to prepare a new campus plan. That document provided some guidance for the post-war development of the campus, including the placement of Bean Hall, Oregon Hall, some science facilities, and an addition to the Knight Library. Two major campus structures built in the late 1960's, the Student Health Center and the Law Center, were not contemplated by the plan, and one of its main features--the development of academic buildings on the Pioneer Cemetery site--was never implemented.

By 1973 the need for a new plan was acknowledged and the Center for Environmental Structure, headed by Christopher Alexander, was retained for that purpose. The Oregon Experiment was the result of this collaboration between the Center and the University. Its principal characteristic is the establishment of a process for making development decisions on an ongoing basis as a replacement for the static (a fixed image) master plan. This concept acknowledges the fact that although change will occur, the exact nature and magnitude of that change cannot be predicted with any degree of certainty, and that object-oriented plans based on explicit assumptions about the future become outdated as that "future" becomes known.

The new Long Range Campus Development Plan is intended to represent a continuation of these planning traditions. A large body of norms, traditions, and development policies has developed over the course of the institution's history, but has heretofore remained unwritten or at best recorded only in repetitive actions of individuals and groups engaged in campus development activities. The intent of a new plan is to unify these norms, traditions, and policies with the essential elements of the Lawrence ideal and the fundamental principles of The Oregon Experiment.

The Oregon Experiment approach was based on the University's desire to develop a planning method that met three goals as it guided growth and change as follows:

1. The solution is a process and not just a map;
2. The process honor and strengthen the University's tradition of meaningful consultation with students, faculty, and staff;
3. The plan provides for continuous adjustment of campus facilities in response to changing educational policies and programs.

The principal characteristic of the Oregon Experiment approach is the establishment of a process for making development decisions on an ongoing basis as a replacement for the static master plan. This concept acknowledges the fact that although change will occur, the exact nature and magnitude of that

change cannot be predicted with any degree of certainty, and that object oriented plans based on explicit assumptions about the future become outdated as that "future" becomes known. The adoption of the Oregon Experiment approach resulted in the establishment of six fundamental principles that guide their planning processes today. These principles are as follow:

1. Organic order: This principle suggests that development of the campus should be guided by explicitly debated and approved basic policies (or "patterns") that articulate shared traditions and understandings of the University community, rather than a "fixed image" master plan (the dreaded map). This principle developed largely in response to the notion that "fixed image" plans could not reflect or adapt to changes in community values, expectations, or goals as effectively as could state policies.

2. Piecemeal growth or continuous adaptation: This principle acknowledges that development of the campus occurs gradually over time and that although there will be needed for large projects from time to time, available funds ought to be distributed in a way that allows for continuous care and improvement of the entire campus.

3. Patterns: Patterns are design statements that describe and analyze the development issues and suggest ways in which those issues might be

resolved. This principle, which is perhaps the most famous, largely because of Alexander's book *A Pattern Language*, calls for the establishment of patterns that articulate commonly held values as they pertain to the campus environment. (Planners often refer to patterns as "policy statements," a term that seems to be more easily understood.)

4. Diagnosis: This principle calls for a periodic analysis of the campus to provide a general context for implementing new projects. Because the University has not been able to provide staff for regular diagnoses, they are performed in conjunction with the early planning stages of new projects.

5. Participation: The virtual cornerstone for the entire planning process is the notion that the people most directly affected by the results of development are best equipped to guide it and should be directly involved in its planning. This concept, which seemed radical in 1975, has now become more of the norms than the exception in planning efforts everywhere.

6. Coordination: This principle recognizes that the University as a whole has a wide range of interests to be considered and that separate development activities must be coordinated. The Campus Planning Committee serves as the coordinating agent. It makes recommendations

to the President on the acceptance of all projects and also serves as advisory to the President and to the Planning Office staff on policy issues relating to transportation, design, and other campus-wide development activities. The Campus Planning Committee has the authority to recommend approval of a project that violates the letter but not the spirit of anyone policy, and thus serves as a sort of continuous monitor of the health of those policies.

The Long Range Campus Development Plan guides the development of individual projects on the campus within the intent of these principles. The document provides for the principle of Organic Order in the way it prescribes planning efforts. It lists the Patterns, which are most important to the campus. Its procedures dictate the use of Participation on individual projects and call for Coordination through the review of all projects by the Campus Planning Committee. Piecemeal Growth and Diagnosis are principles that each individual project must accommodate.

Because of the modest size of the campus, the planning efforts are mostly spent on the implementation of specific projects rather than the refinement of the Long Range Plan itself. The management of the process takes some effort, but the results—both in terms of the flexibility gained and the buildings produced—to be worthwhile.

Figure 33 shows the master plan of the university that reflects the impacts of

Figure 33: The master plan of the university of Oregon. ¹²⁸

ideas and process of design as described and defined by Alexander. Is it very different from any other university campus? Why this experience failed? I will talk about this later in the last section of this appendix.

A Pattern Language: Towns, Buildings, Construction.

Alexander's "Pattern Language" describes a highly structured collection of patterns intended as a practical guide for architects. ¹²⁹ The language contains 253 patterns split into three broad categories, towns, buildings and construction.

Towns (94 patterns): These patterns cover far more than just urban planning and examining the responsibility of the designer from a global perspective. Alexander suggests that the designer should think about the issues which surround every day and political life in a wider context when embarking on any new project. This clearly illustrating the importance which Alexander puts on the work of the architect in shaping society. By defining architecture, urban planning and design as one discipline, he rejects the North American definition of the architectural profession.

Alexander also examines the role of the designer in the development of urban and rural regions within the borders of a nation. By examining both the urban spaces which the architect creates, and the rural spaces, Alexander highlights the tension between urban and rural needs caused by continued development.

However, the book is mainly concerned with the urban regions and having established the framework in which town development is situated.

Alexander moves on to consider the structure of towns and cities. The

decomposition of spaces continues through the examination of various services that must be provided public transport systems, roads, common land, as well as the arrangement of industrial and residential areas.

This group of patterns is influenced by studies in human behavioral sciences and tries to address many of the issues which people have to face living in a modern city. The result is a prescription for cities to develop naturally according to the needs of the population. It proposes that city planning should not completely dictate the nature of each area but facilitate the development and growth of local communities and neighborhoods. It promotes the importance of local communities through the provision of services and facilities.

Buildings (100 patterns): In this set of patterns, Alexander concerns himself primarily with the specification of the family home and then the workplaces, and factory environments. Some of the patterns included for the home could also be used in a more general context. The pattern "Intimacy Gradient" encourages that homes are designed with a series of levels of privacy radiating away from public entrances and main public spaces. This pattern could be adapted to the design of any inner and outer space in any context.

Alexander makes great use of the human sciences to derive patterns for buildings in the same way that he did when considering the importance of communities and their role in urban planning. In the design of a building Alexander draws upon psychological studies in particular and suggests patterns for the provision of facilities that these studies recommend. The need for privacy is recognized both for couples, individuals. Also, the need for communal activities to reassert bonds between the members of a family is recognized. This is an attempt by Alexander to modernize the patterns that are derived from an older "family formation pattern" and to adopt them with new forms of a family lifestyle.

However, Alexander does stray from the course of defining buildings to defining the lifestyle of those who are to occupy them. In doing so he begins to prescribe how the activities of occupants should be conducted. As an acceptable example Alexander encourages the eating of communal meals together as an important part of family life and introduces some patterns for the provision of suitable facilities within a building for this purpose. "Bathing Room" and "Bed Cluster" are examples of unacceptable suggestions. "Bathing Room," advocates the provision of communal bathing facilities for the whole family to bathe together. Culturally this may not be acceptable to many people because of their social or religious

upbringing. It is debatable whether this type of prescription should be within a language for building design. Although, obviously, if such a facility were to be provided it would have many repercussions throughout the process of design.

"Bed Cluster" is similar to a pattern that prescribes a lifestyle that many may find unacceptable. Here, Alexander advocates that all members of a family should sleep in a single room as an important point of contact between the family members. Although this may be a well-founded belief that all members of a family should sleep close to each other, it is an idea that is foreign to many people in Western societies. It is also difficult to see how to resolve this pattern with others where Alexander advocates privacy for couples and individuals: "Couple's Realm" and "Teenager's Cottage." In an admission of the practical constraints imposed upon a designer there is some redundancy in the scope of the patterns. This is recognition of the fact that not all design patterns can be considered within the scope of a single project. By introducing a rich set of overlapping patterns, beneficial properties are reinforced, increasing their likelihood of consideration in a particular design.

The three patterns, "Wings of Light," "Long Thin House" and "Light on Two Sides" are examples of overlapping. Although they obviously all refer to

different aspects of building design and are correctly separated as design patterns, there is an area of overlap between them. That is, both "Wings of Light" and "Long Thin House" advocate restricting the width of spaces within a building in part to ensure good lighting across the area. Similarly, "Light on Two Sides" also advocates good lighting within the enclosed spaces of a building by providing more than one natural light source to any room, reducing the stark contrasts inherent in rooms lit by a single window.

Of course if the advice in "Long Thin House" was always followed, it would be somewhat redundant to include a pattern for "Wings of Light."

Additionally, if a certain design of the interior spaces could be guaranteed to be followed within a building's construction, it would be a trivial matter to specify the lighting requirements for the space within the same pattern.

Obviously, it is not possible in practical design problems to rigidly follow a set of patterns and the pattern language recognizes this and repeats similar messages throughout the book with various levels of specificity.

With a language as complex as this, one would expect to find a certain amount of ambiguity of appropriateness to a design problem and contradiction between the patterns. It is a tribute to the construction of the language that it is difficult to find such contradictions. To some extent the

problem of contradictions is eased by the redundancy within the language. However, the provision of strictly redundant patterns does increase the difficulties in selecting a set of appropriate design patterns when faced with a problem. This raises the question of whether it is possible to create a normative language of design which avoids both contradictions and ambiguity, while still remaining rich enough to be useful for the complex problems, faced in practice.

Construction (59 Patterns): Having provided a definition for good buildings, Alexander finishes his collection of patterns by defining the right way to construct buildings.

As might be expected, these patterns are the most specific in their recommendations. They specify classes of materials and approximate dimensions appropriate for various construction tasks.

Many of the patterns take historical cues from proven successful designs as a starting point for further development. For example the advice to use columns to delineate spaces, can be used in various ways to add interesting features to both internal and external spaces.

The Timeless Way of Building

"But in our time the languages have broken down. Since they are no longer shared, the processes which keep them deep have broken down. It is therefore

virtually impossible for anybody, in our time, to make a building live."

"The Timeless Way of Building" and "A Pattern Language: Towns, Buildings, Construction" were written as a pair, with the former presenting rationale and method, and the latter concrete details.¹³⁰ They present a fresh alternative to the use of standardized models and components and accentuate the philosophical, technical and social-impact differences between analytic methods and the adaptive, open, and reflective (all in several senses) approach to design that Alexander is reaching for.

The term pattern is a performed construct, which Alexander does not define. This term describes sets of forces in the world and relations among them. In *Timeless*, Alexander describes common, sometimes even universal patterns of space, of events, of human existence, ranging across all levels of granularity. *Timeless Way of Building* is the essence of Alexander's theory and the rest of his books and experiences focused on testing and elaborating the concepts that were introduced in this book. These concepts are as follow:

1. ***Timeless Way of Building***: A building or town will only be alive to the extent that it is governed by the timeless way. *Timeless Way of Building* is a process that brings order out of nothing but us; it cannot be attained, but it will happen of its own accord, if we will only let it. To seek the timeless way we must first know the quality without a name.

2. *The quality without a name:* Quality without a name is a central quality that is the root criterion of life and spirit in a man, a town, a building, or a wilderness. This quality is objective and precise but it cannot be named. The search that we make for this quality, in our own lives, is the central search of any person and the crux of any individual person's story. It is the search for those moments and situations when we are most alive. In order to define this quality in buildings and in towns, we must begin by understanding that every place is given its character by certain patterns of events that keep on happening there.

3. *The patterns:* These patterns of events are always interlocked with certain geometric patterns in the space. Indeed, as we will see, each building and each town is ultimately made out of these patterns in space and out of nothing else: they are the atoms and the molecules from which a building or a town is made. The specific patterns out of which a building or a town is made may be alive or dead. To the extent they are alive, they let our inner forces loose and set us free, but when they are dead, they keep us locked in inner conflict. The more living patterns there are in a place—a room, a building, or a town—the more it comes to life as an entity, the more it glows, the more it has that self-maintaining fire which is the quality without a name. When a building has this fire then it becomes a

part of nature. Like ocean waves, or blades of grass, its parts are governed by the endless play of repetition and variety created in the presence of the fact that all things pass. This is the quality itself. To reach the quality without a name we must then build a living pattern language as a gate.

4. *The Gate:* This quality in buildings and in towns cannot be made but only generated, indirectly, by the ordinary actions of the people, just as a flower cannot be made but only generated from the seed. The people can shape buildings for themselves and have done it for centuries by using pattern languages. A pattern language gives a user the power to create an infinite variety of new and unique buildings just as his ordinary language gives him the power to create an infinite variety of sentences. These pattern languages are not confined to villages and farm society. All acts of building are governed by a pattern language or some sort, and the patterns in the world are there, entirely because they are created by the pattern language which the people use. It is not just the shape of the towns and buildings which come from pattern languages. It is their quality as well. Even the life and beauty of the most awe-inspiring great religious buildings came from the languages their builders used. In our time the languages have broken down. Since they are no longer shared, the

processes which keep them deep, have broken down and it is therefore virtually impossible for anybody, in our time, to make a building live. To work our way toward a shared and living language once again we must first learn how to discover patterns which are deep and capable of generating life. We may then gradually improve these patterns which we share by testing them against experience. We can determine, very simply, whether these patterns make our surroundings live, or not, by recognizing how they make us feel. Once we have understood how to discover individual patterns which are alive we may then make a language for ourselves for any building task we face. The structure of the language is created by the network of connections among individual patterns: and the language lives, or not, as a totality to the degree these patterns form a whole. Then finally, from separate languages for different building tasks we can create a larger structure still, a structure of structures, evolving constantly which is the common language for a town. This is the gate. Once we have built the gate, we can pass through it to the practice of the timeless way.

5. *The Way:* Now we will begin to see in detail how the rich and complex order of a town can grow from thousands of creative acts. For once we have a common pattern language in our town, we will all have the power

to make our streets and buildings live through our most ordinary acts. The language, like a seed, is the genetic system which gives our millions of small acts the power to form a whole. Within this process, every individual act of building is a process in which space gets differentiated. It is not a process of addition, in which pre formed parts are combined to create a whole, but a process of unfolding, like the evolution of an embryo, in which the whole precedes the parts, and actually gives birth to them, by splitting. The process of unfolding goes step by step, one pattern at a time. Each step brings just one pattern to life and the intensity of the result depends on the intensity of each one of these individual steps. Shaping one building from a sequence of these individual patterns, whole buildings with the character of nature will form themselves within your thoughts, as easily as sentences. In the same way, groups of people can conceive their larger public buildings on the ground, by following a common language, almost as if they had a single mind. Once the buildings are conceived like this, they can be built, directly, from a few simple marks made in the ground again within a common language, but directly and without the use of drawings. Next, several acts of building, each one done to repair and magnify the product of the previous acts, will slowly generate a larger and more complex whole than any single act can generate. Finally, within the

framework of a common language, millions of individual acts of building will together generate a town which is alive, and whole, and unpredictable, without control. This is the slow emergence of the quality without a name, as if from nothing. As the whole emerges, we will see it take that ageless character which gives the timeless way its name. This character is a specific, morphological character, sharp and precise which must come into being any time a building or town becomes alive: it is the physical embodiment, in buildings, of the quality without a name. Yet the timeless way is not complete, and will not fully generate the quality without a name, until we leave the gate behind.

6. *The kernel of the way:* Indeed this ageless character has nothing, in the end, to do with languages. The language, and the processes which stem from it, merely release the fundamental order which is native to us. They do not teach us, they only remind us of what we know already, and of what we will discover time and time again, when we give up our ideas and opinions, and do exactly what emerges from ourselves.

A New Theory of Urban Design

The venerable cities of the past, such as Venice or Amsterdam, convey a feeling of wholeness, an organic unity that surfaces in every detail, large and small, in restaurants, shops, public gardens and even in balconies and

ornaments. But this sense of wholeness is lacking in modern urban design with architects absorbed in problems of individual structures and with city planners preoccupied with local ordinances. This fact makes it almost impossible to achieve. ¹³¹

In this book Christopher Alexander presents a new theory of urban design which attempts to recapture the process by which cities develop organically. To discover the kinds of laws needed to create a growing whole in a city, Alexander proposes here a preliminary set of seven rules which embody the process at a practical level and which are consistent with the day-to-day demands of urban development.

He then puts these rules to the test, setting out with a number of his graduate students to simulate the urban redesign of a high-density part of San Francisco. They initiate a project that encompassed some ninety different design problems, including warehouses, hotels, fishing piers, a music hall, and a public square. This extensive experiment is documented project by project, with detailed discussion of how each project satisfied the seven rules. It was also, accompanied by floor plans, elevations, street grids, axonometric diagrams and photographs of the scaled-down model which clearly illustrate the discussion.

A New Theory of Urban Design provides an entirely new theoretical framework for the discussion of urban problems, one that goes far to remedy the

defects which cities have today.

A Foreshadowing of 21st Century Art: the Color and Geometry of very early Turkish Carpets

In this book Alexander connects geometrical patterns with aspects of physical reality. He provides a solid connection that links design and the visual environment, on the one hand, with beauty and living forms on the other. ¹³²

Alexander has obtained a solution following his long study of oriental rugs to the central problem of recognition in art: "What makes us recognize or identify a bird, a fish, writing, or a leaf, when what we see does not bear any real resemblance to what we call it?" [Grabar, 92]

This question also relates to how one can quantify beauty. Alexander has probed into this 'forbidden' topic and discovered that in order to make beautiful buildings you have to know how to make beautiful objects of whatever size. Further, he has discovered rules for making beautiful things from studying the patterns on very old carpets. These rules are universal and they help to make anything beautiful, from a doll to an apartment complex.

This result is entirely independent from its source, which are the old carpets. Alexander outlines the rules for making beautiful things in the first quarter of his book. For Alexander the basic building block of any design is a 'center'. This is a perceptible region of coherent space. For example, a dot in the center of a page

holds the page together, as does a circle or square in a page. Adjoining and overlapping centers can either reinforce each other or detract from each other, and they do not need to be symmetric. Centers also interact at a distance. The best patterns or designs are simply those which can be decomposed into the maximum number of mutually reinforcing centers.

Alexander illustrates this process by reference to the carpet designs, using sketches that isolate the various centers. For example, the "small pattern Holbein," has hundreds of different centers, and all densely packed. Even a superficially simple design, if it evokes profound feelings, turns out to contain an enormous number of mutually reinforcing centers. Of course, one does not ordinarily decompose a carpet into its "field of centers" in order to judge how attractive it is. It happens, however, that in side-by-side comparisons of two carpets or any other similar objects, the one with the more intense field of centers is invariably the one which is most attractive to most people.

Alexander compares any two objects and asks which of the two most resembles one's own inner self -- one's character or one's soul. Usually, there is a clear choice and the one chosen is that which contains the most intense field of centers. This 'mirror of the self' test makes use of basic instincts that humans already have about the field of centers. Showing two similar rugs to novices and asking them to choose according to the 'mirror of the self' test, they usually

choose the most valuable one. They may prefer the brighter colors or thicker wools pile of the other but are, nevertheless, forced to choose the one that most resembles their conception of self.

There is a further and inescapable conclusion. If one constructs inanimate objects with a very intense field of centers, they begin to resemble living things. This result is surprising. If one applies the field of centers' analysis to any living object, animal or plant, then one is overwhelmed by its intensity. Therefore, the rules for creating beautiful things lead to inanimate objects that have the subconscious feeling of living things without necessarily looking like any living thing at all. And here is the answer to the question on representation via abstract forms. An abstract drawing by a master who intuitively creates a dense field of centers can convey more of the essential sense of a living thing than a photograph.

According to Alexander, contemporary art and architecture contain almost none of the beauty that is found in old carpets. He believes that the aesthetic rules apply to all things, and most certainly to all types of carpets. Alexander's personal preference for "very early Turkish carpets," because he finds in them the most intense field of centers, should not detract from the universal applicability of his rules. These rules illustrate unequivocally why carpets can indeed be great works of art.

In this book Alexander describes how "truth" and "beauty" are present in ancient Turkish prayer carpets. The artisans who created these carpets were striving to capture the likeness/face of God in them, the structure of the universe. Alexander, through his studies, has moved out of culturally sensitive patterns into the cognitive and psychological aspects of geometry, which are universal. He is concerned with unlocking the secret behind what constitutes the essence of what he now refers to as "wholeness," or that which makes some carpets more "whole" than others.

Ian Bennett [Bennett, 95] in his review of this book concentrates mainly on his many technical disagreements with Alexander. While these concerns are important to oriental carpet enthusiasts, they draw attention away from Alexander's central contribution: a set of tools to judge beauty. These can enhance one's appreciation of art and its relation to life in general. This book should be read by a wide variety of people interested in what makes things beautiful and alive. Bennett's review implies that Alexander's theory of centers and internal unity is somehow already contained in historical writing on European art. He also claims that Alexander's rules fail to apply to carpets outside the present collection. Both of these conclusions are wrong. There are no references to patterns and their configuration rules and value systems after Renaissance in cultural or artistic documents. Scientific references to patterns are not dated

earlier than the beginning of this century.

Alexander's pattern theory deeps its roots in a forgotten language. This theory is a great effort to translate this forgotten language to the modern language of systems. Alexander found an untouched part of pattern languages in the carpet design. I briefly look at the carpet literature to show the reason of Alexander's special interest in this artifact.

The Language of the Carpets

No one knows exactly when and where the first knotted-pile carpets were woven, however, the oldest "surviving" pile carpet was discovered in the grave of a Scythian prince in the Pazyryk valley of the Altai mountains (Siberia) by a Russian archeologist (Rudenko) in 1947.¹³³ This carpet is presently displayed in the Hermitage Museum in Leningrad. The carpet was woven with the double knot and contains a surprising 347,000 knots per square meter (255 per square inch); it is 3.62 square meters (6 x 6.5 feet) and has been carbon dated to have been from the 5th. Century B.C. It was loaded and subsequently flooded and froze to await discovery by Rudenko. The Pazyryk, or Altai carpet, is rather sophisticated, thereby showing that it is the product of a long history and tradition of weaving. Through this long history carpet design has developed a symbolic language in parallel to the other fields of art and culture.

A carpet is a communicative statement written in its symbolic language.

Figure 34 shows some words of the local version (Turkish) of that symbolic language of carpet design. A concept oriented pattern language would be used to create a carpet, using these words.

Figure 34: The Language of Carpets: words (part 1)

<p>Status Hair Band: (Single) This sign expresses the yearning of a young woman to get married. Traditionally in Anatolian villages the girls keep their hair long and will not cut it until they get married.</p>	<p>Ying & Yang: This motif signifies that the weaver is married as well as love and unity. Inherited from the Far East, this symbol denotes love and unity between a man and women. A dot of the opposite color in each half shows that nothing is pure in nature.</p>	<p>Hands on Hips: The mother Goddess of ancient Matriarchal beliefs. At an early stage goddesses represented all superhuman powers. This motif is only shown when the weaver gives a birth to a boy. The hands on hips show that she is very proud.</p>
<p>Happiness: Motifs such as evil eye and ram's horn signify that the weaver is happy and she is thanking of God for her happiness.</p>	<p>Family Sign: Family, or clan signs are used all tribal people, to mark their sheep, kilims and other possessions.</p>	
<p>Eye (Evil Eye): This signifies a bad, or nasty look, which is believed to be encountered by an object, which looks similar to an eye. In its most simplified form a triangle is used.</p>	<p>Ram's Horn: The ram's horn denotes fertility, heroism and power.</p>

Figure 34: *Language of carpets: words* (part 2)

Relationship: The chest, comb, fetter and fertility motifs express a weaver's relationship with her husband and her in-loves.		
<p>Chest Comb: The chest and comb motifs are symbols of the bridge, marriage and happiness in Anatolian folklore. The chest, or clothes sack among wandering tribes, represent the girl's longing for marriage, since they contain her trousseau or dowry.</p>	<p>Fertility: The relationship between the sexes, and proliferation. Stylized versions of multigrain plants, for example, wheat and pomegranate that denote fertility.</p>	<p>Fetter: A fetter is used to prevent horses from running away. In kilims, it represents harmony and togetherness of lovers.</p>
<p>Birds: Bird motifs have various meanings. Birds of prey, such as eagle falcon and hawk represent strength and power. These bird symbols can be found on the Selcuks and Ottomans. Birds can also symbolize the celestial messenger and longevity. The phoenix and the dragon fighting symbolize the coming of spring rain.</p>	<p>Animals: From the earliest times people have believed that by imitating, or weaving part of a dangerous animal, they will have power over it and protection from it. Carrying a wolf's paws, a crocodile's tooth or a dried snake or scorpion is remnant of this ancient belief, which is still practiced in some places today.</p> <div> <p>Scorpion</p> </div> <div> <p>Wolf's Print</p> </div> <div> <p>Dragon</p> </div>	

Domestic Architecture

"Doors of Perception" is the title of a long-term investigation of interactivity and design at the Netherlands Design Institute in Amsterdam.¹³⁴ The Doors of Perception's annual conference is co-produced by the Institute and Mediamatic. The Design Institute's interactive design program started off when the Institute and Mediamatic organized the first conference in October 1993, and continued with the second edition in November 1994. Christopher Alexander was one of the speakers of "Doors of Perception 2@HOME Conference" during 4-6 November 1994 in Amsterdam. His speech titled "Domestic Architecture," was electronically published as a part of the documents of that conference. In this speech Alexander talks about one of his most important ideas: "User participation." He had explained one of his experiences in working with people. He concluded that:

"In my work, I've become aware that progress toward that blissful state cannot be accomplished except by a large and conscious effort by people themselves, interacting with their environment through time. The two major things I've mentioned are the participation of people in their own house design and a drastic modification of construction procedures. But the kind of process that I'm talking about now is much more drastic than that. To create that kind of process is a huge undertaking. After 20 years work, even though I have achieved something, I feel I'm really only at the

beginning of a very different kind of journey. One aimed at finding the ways that more conscious process is taking place. One of the ways I've begun to explore it is simply through painting. Trying to paint the kind of world in which this blissful state might or can exist. "

The Nature of Order

This book has been in preparation for more than thirty years, and encapsulates all of Christopher Alexander's theories. Oxford University Press will publish the book during 1999. [Latest information, May 1999] ¹³⁶

All the information that is included in this part is extracted from the notes of the people who had read its draft. In the present format this book has four large volumes, a prolog and an epilogue titled as follow:

1. **Prologue: The Art of Building and the Nature of the Universe.**
2. **Book One: The Phenomenon of Life.**
3. **Book Two: The Process of Creating Life.**
4. **Book Three: A Vision of a Living World.**
5. **Book Four: The Luminous Ground.**
6. **Epilogue: The Art of Building.**

The first two volumes of this book should be considered as a new approach to understanding structure, both natural and man-made. They span aesthetics, science, and architecture, and are relevant to any complex process. Volume three is of immediate interest to architects and urban planners as it contains a large number of examples of the building process from the largest to the smallest scale. Volume four of the Nature of Order is a deeply spiritual work. The last of the four volumes transcends architecture, and plunges into what connect us with our universe. It reveals how superficial our century has been in addressing the fundamental qualities and needs of human beings. This volume promises to have a profound impact on our society, even our civilization. For that reason, it might ultimately be the most revolutionary aspect of the entire work.

In this book, Alexander develops a comprehensive theory of how matter comes together to form coherent structures. Paralleling, but not copying, recent results from complexity theory, he argues that the same laws apply to all structures in the universe: atoms, crystals, living forms, and galaxies. Human beings apparently have a built-in (though subconscious) understanding of these laws. Man's creations have the option of following the same laws, or violating them. Those that follow them result in our greatest achievements, either as artifacts, as buildings, or as cities.

Far-reaching results on urban planning were given some years ago in Alexander's famous article "A City is not a Tree," [Alexander, 92] and in the "Pattern Language" [Alexander, et al., 77] and "A New Theory of Urban Design." [Alexander, et al., 86] His solutions were abstracted from and checked against urban sites that work, in order to prove their sufficiency in applying them to urban sites that don't. He goes much further in *The Nature of Order*, showing that the rules governing the growth of neighborhoods and cities arise from a fundamental law of nature. Alexander has discovered the process that governs the growth of a successful city - which is the same process by which organic and inorganic forms evolve. Surprisingly, the law concerns the *process*, not the form or plan. This whole approach might seem unfamiliar to urban planners who think in terms of static images, though biologists will immediately recognize it as correct. These universal laws apply not only to "traditional" cities; they apply to all cities, in every age and in every culture that enhance human activity.

Through the following paragraphs that are extracted from Book Four of *The Nature of Order*, Alexander describes his new hypothesis as follows:

"The structure of life I have described in buildings – the structure which I believe to be objective - is deeply and inextricably connected with the human person, and with the innermost nature of human feeling. In this fourth volume I shall approach this topic of the inner feeling in a building,

*where there is a kind of personal thickness — a source, or ground,
 Something almost occult - in which we find that the ultimate questions of
 architecture and art concern some connection of incalculable depth,
 between the made work (building, painting, ornament, street) and the
 inner *œ* which each of us experiences.*

*What I call "the *œ* " is that interior element in a work of art, which makes
 one feel related to it. It may occur in a leaf, or in a picture, in a house, in a
 wave, even in a grain of sand, or in an ornament. It is not ego. It is not me.
 It is not individual at all, having to do with me, or you. It is humble, and
 enormous: that thing in common which each one of us has in us. It is the
 spirit which animates each living center.*

*I believe that the ultimate effort of all serious art, is to be making things
 which connect with this *œ* of the person. This *œ*, not normally available,
 is dredged up, forced to the light, forced into the light of day, by the work
 of art.*

*My hypothesis is this. That all value depends on a structure in which each
 center, the life of each center, approaches this simple, forgotten,
 remembered, unremembered *œ*. That in the living work, each living
 center really is a connection to this *œ*.*

I believe that this is true; not just a nice way of talking. As I try to explain it, quietly for all its grandeur, and try to make the artist's experience real, I hope that you, with me, will also catch a glimpse of a modified picture of the universe.

For I believe it is the nature of matter itself, which is soaked through with $\mathcal{S}$. The essence of my argument in Book 4 is that the $\mathcal{S}$, the thing I call the $\mathcal{S}$, which lies at the core of our experience, is a real thing, existing in all matter, beyond ourselves, and that we must understand it this way in order to make sense of living structure, of buildings, of art, and of our place in the world.

That very difficult intellectual path, is the path which lies before me. I shall try to persuade the reader that this is literally true".

This book promises to be "a new paradigm for programming." It is written primarily in the interest of architects (of buildings, not software). Since the book is not publicly available, I used the notes of those people who had access to the drafts of the book to give a brief overview of the book.

An overview of this book can be found in Richard Gabriel's book: *Patterns of Software*. [Gabriel, 96] There are notes of a review talk given by Jim Coplien on the *Nature of Order*, [Coplien, 97-2] with commentary by Brad Appleton. Jim Coplien is

writing a series of articles in C++ Report outlining the possible applications of the *Nature of Order* to software.

In this book, as Coplien has reported, [Coplien, 97-2] Alexander is claiming that beauty is mathematical. This Platonic idea cannot be supported by the heritage of pattern-based architecture.

*A Proto Pattern Language for Human-Computer Interaction***Appendix 2: Section 3**

Evaluation of Alexander's Idea

A Proto Pattern Language for Human-Computer Interaction

Appendix 2: Section 3:

Evaluation of Alexander's Idea

Christopher Alexander's help found the UC Berkeley Center for Environmental Studies in the late Sixties. ¹³⁶ In 1978, a major statement about design in the form of three books was announced by this center. In the words of writers of these three publications, it was supposed that these books "lay the basis for an entirely new approach to architecture, building, and planning, which we hope will replace existing ideas and practices entirely." ¹³⁷

The resulting trilogy, *The Timeless Way of Building (the manifest)*, *A Pattern Language (the technique)*, and *The Oregon Experiment (the application)* formed a new theory of design. Alexander's theory is an attempt to "rationalize" the creative artistic process of architecture. Therefore, his approach is not popular in the non-institutional architectural practice which insists on the irrational nature of the artistic process. Instead, the concept of patterns and pattern languages, and the principle of user participation which are inspired by Alexander's ideas, became popular in the software engineering community, and in the field of urban planning and design.

Vincent Scully, in *Modern Architecture* refers to Alexander's Pattern Theory

as: [Scully, 74] *"Slightly different is the American attempt to computerize design, which is best associated with the researches of Christopher Alexander and his teaching at Berkeley. Like Buckminster Fuller's geodesic dome it is indeed very American, seeking a method technologically sophisticated, very clear, promising mechanical answers. Its deterministic purism is sympathetic too much American thought, but it hardly satisfies all American objectives..... The act of building one of these mechanisms is more than a mechanical experiment for Fuller and his followers; it is a mystical rite, to be repeated with little variation over and over again, eternally wonderful, always complete."*

Alexander's approach to design in Notes on the Synthesis of Form was very mechanical. According to Lea, [Lea, 93] "Notes was an expression of a more innocent era in engineering and analysis, built upon the essential optimism of a host of postwar intellectual movements: cybernetics, systems analysis, operations research, and computing." To Rohit Khare Alexander's method is less robotics than this passage by Lea makes it sound; "it is conceptual kin to Louis Kahn's theories of "basic-concept -deformed-by-contextual-special-cases."
[Khare, 95]

Alexander criticizes Modern design because of the five following reasons:

- 1- Inability to balance individual, group, societal, and ecological needs.
- 2- Lack of purpose, order, and human scale.

3- Aesthetic and functional failure in adapting to local physical and social environments.

4- Development of materials and standardized components that are ill suited for use in any specific application.

5- Creation of artifacts that people does not like it.

Alexander and his colleagues, inspired by the image of the European village and their fieldwork in Peru and India, advocate the traditional, indigenous design over "Main Street" notion. They actually follow "Berkeley consciousness of social justice and impending late '70s apocalypse." Their approach is a rediscovery and systematization of the most ancient design theory: *analogy*.

In "A Pattern Language," Alexander presents 253 problem-solving-processes from the scale of cities and cultures to construction and decoration. It is not an Erector set of compatible "design modules" any more than a dictionary is a toolkit. Indeed, its flexibility and possibility justify its denotation as a Language. Each pattern is a part of the vocabulary of the Alexandrian world, structured by a grammar relating patterns across scales.

Alexander, through his studies, has moved out of culturally sensitive patterns into the cognitive and psychological aspects of geometry, which he considers universal. He is concerned with unlocking the secret behind what constitutes the

essence of "wholeness." Through empirical results of his experiments, Alexander has arrived at fifteen fundamental properties, which according to him form humans' most fundamental notions of order and beauty. Some of these may be even deeper, relating to survival instincts and evolution. These five fundamental structural properties include:

- 1. Levels of Scale:** within or around a given center, other centers should be within a factor of four to a factor of 10 of the same size. (**A mechanical view of proportion**) ¹³⁸
- 2. Strong Centers:** centers shouldn't overlap too much; they should have their own identity. (**Identity**)
- 3. Alternating Repetition (Rhythm)**
- 4. Boundaries:** each center gives rise to fuzzy centers that enclose the center itself, forming rather wide boundaries (**Freedom of boundaries**)
- 5. Positive Space:** the space waiting to become a center; a latent center (**Positive space**)
- 6. Good Shape:** there are simple shapes that are pleasing to the eye (triangles, circles, stars.) (**Generic forms**)
- 7. Local Symmetries (Symmetry)**

8. Deep Interlock and Ambiguity: like city-country fingers in. (**Integrity and absorption**)

9. Contrast (Contrast)

10. Graded Variation: step-by-step gradients pointing toward the center and strengthening it. (**Continuity**)

11. Roughness: particularly from centers that bump into each other as the system unfolds. (**Purity**)

12. Echoes: repeated structural similarities. (**Rhythm**)

13. The Void: periodic cleaning out or self-organizing that preserves overall structure by cleaning out overly intense centers. (**Purification**)

14. Simplicity and Inner Calm: "It is essential that the great beauty and intricacy of ornament only just far enough to bring this calm into being, and not so far that it destroys it." (**Calmness**)

15. Not-Separateness: eliminate exaggerated difference; unification (**Integrity**)

Alexander's process for applying patterns can be iterative or recursive, but the recursive perspective corresponds more to the geometric nature of his

pattern language. Each pattern works with a finer structure than the one before it, preserving the overall structure of the whole. The result is fractal in nature.

Alexander's later work, more consciously aspires to this fractal nature.

Alexander's "rug experiment" and much of his earlier research explored this phenomenon of objective beauty. He concluded that beauty owes strongly to geometric properties. He developed a theory of beauty that transcends architecture, other human crafts, and structures in nature, based on centers, structural features, and process. This Platonic theory is different from the definition of beauty in the pattern based architecture of beauty. The new-platonic view of pattern-based architecture considers geometry as a tool, not as the essence of the beauty.

*A Proto Pattern Language for Human-Computer Interaction***Appendix 3:**

Patterns in Software

Software Patterns
Historical Process

*A Proto Pattern Language for Human-Computer Interaction***Appendix 3:**

Patterns in Software

Software Patterns

Software patterns have emerged from the object-oriented community. Software patterns are a literary form of software engineering problem-solving that have their roots in a design movement of the same name in contemporary architecture, literate programming, and the documentation of best practices and lessons learned in all vocations. ¹³⁰

Any science or engineering discipline has a common vocabulary for expressing its concepts and a language for relating them together. The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of the software development. Software patterns help create a shared language for communicating insight and experience about these problems and their solutions. Formally codifying these solutions and their relationships lets software designers successfully capture the body of knowledge which defines their understanding of

good architectures. Forming a common pattern language for conveying the structures and mechanisms of software architectures allows designers to intelligibly reason about them. The primary focus is not so much on technology as it is on creating a culture to document and support sound engineering architecture and design.

Software patterns first became popular with the publication of "Design Patterns: Elements of Reusable Object-Oriented Software. [Gamma, et.al., 95] Patterns have been used for many different domains ranging from organizations and processes to teaching and architecture. At present, the software community is using patterns largely for software architecture, design, development processes and organizations. Other recent books that have helped popularize patterns are Pattern Languages of Program Design volumes 1 to 4 [Coplien, et.al., 95], [Vissides, et.al., 96], [Martin, et.al., 97], [Harrison, et.al., 99] which were selected papers from the first and second conferences on Pattern Languages of Program Design (PLOP). Many of these books are part of the Software Patterns Series from Addison-Wesley.

The current use of the term "pattern" in software design is derived from the writings of the architect Christopher Alexander who has written several books on the topic as it relates to urban planning and building architecture.

Historical Process

1987: Ward Cunningham and Kent Beck during consulting with " Tektronix's Semiconductor Test Systems Group " decided to try out the pattern to solve some problems in design. Using Alexander's ideas, Ward and Kent let representatives of the users (a trainer and a field engineer) finish the design. They came up with a five pattern "language" that helped the novice designers take advantage of Smalltalk's strengths and avoid its weaknesses:

WindowPerTask, FewPanels, StandardPanels, NounsAndVerbs, ShortMenus.

Ward Cunningham and Kent Beck were amazed at the elegance of the interface their users designed. They reported the results of this experiment at OOPSLA 87 in Orlando. They wrote up a panel position and presented at Norm Kerth's workshop on: "Where do objects come from?" Erich Gamma, who was writing and reflecting about object-oriented design in ET++ as part of his Ph.D. thesis, had realized that recurring design structures or patterns were important. The question really was how do you capture and communicate them.

1990: Bruce Anderson gave a talk at Tools90 at which Erich Gamma was present. Bruce Anderson gave a paper at EcoopOpsla90 (Ottawa) and ran a BOF called " Toward an Architecture Handbook " where he, Erich Gamma, Richard Helm, and others got into discussions about patterns. They realized that they have some shared common understandings about the key ideas behind

writing reusable object oriented software.

1991: Just prior to ECOOP'91 Erich Gamma and Richard Helm put together the very humble beginnings of the catalog of patterns that would eventually become Design Patterns. There they identified many patterns including such familiar and unfamiliar patterns as: *Composite, Decider, Observer, and Constrainer*. Many of these patterns made it into Design Patterns; many others remain rough and unpublished to this day. Bruce Anderson ran the OOPSLA workshop. Coincidentally, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides were all there. They would later become the "Gang of Four" that authored the Design Patterns book. Most of the Hillsiders-to-be were there: Ward and Kent, Desmond De Souza, and Norm Kerth, Doug Lea and Wolfgang Pree. Jim Coplien had been cataloguing language-specific C++ patterns he called idioms, which are one of the Kinds Of Patterns. Early manuscripts of this book (early 1989) were used to teach objects and C++ inside AT&T. Addison-Wesley published the book in September 1991.

1992: Bruce Anderson repeated the workshop in 1992, which is where the Gang Of Four properly got together. Frank Bushman's first publication on patterns was presented there as well. Peter Coad had been exploring patterns in parallel as well; he mentioned them in a 1991 issue of his newsletter, and published an article in CACM in 1992.

1993: In May of 1993, at the Thorn wood Workshop at IBM, some of the people who were involved in pattern discussions taught the IBMers how to use objects. Reflective thinking was a big part of this workshop, both for the standard curriculum, and for the extracurricular patterns discussions that took place after hours. Desmond De Souza, Doug Lea, Kent Beck, Ralph Johnson, Bruce Anderson, Ron Casselman, and John Vlissides were the facilitators. In August of 1993, Kent Beck and Grady Booch sponsored a mountain retreat in Colorado where a group (later called: *Hillside Group*) converged on foundations for software patterns. Ward Cunningham, Ralph Johnson, Ken Auer, HalHilde brand, Grady Booch, Kent Beck and Jim Coplien struggled with Alexander's ideas and our own experiences to forge a marriage of objects and patterns. They agreed that they were ready to build on Erich Gamma's foundation work studying object-oriented patterns and to use patterns in a generative way in the sense that Christopher Alexander uses patterns for urban planning and building architecture. They then used the term ``generative" to mean ``creational" to distinguish them from `` Gamma patterns " that captured observations. Bruce Anderson again held his workshop at OOPSLA '93, this time with patterns in the workshop title and prominently on the agenda.

1994-1995: The Hillside Group met again in April 1994 to "plan" the first PLoP conference. Richard Gabriel was also there and he exhorted them to go

into PLoP with confidence. On August 4, about 80 people came together at the Allerton Park estate near Monticello, Illinois, to do just that.

Ward Cunningham and Ralph Johnson were program and conference chairs, respectively. The PLoP proceedings came out in May 1995, as "Pattern Languages of Program Design." In the mean time, the Gang Of Four had wrapped up their work and sent it to the publisher. The first major compendium of patterns between two covers, "Design Patterns: Elements of Reusable Object-Oriented Software" made it out in time for OOPSLA '94. It sold 750 copies at the conference (more than seven times the highest number of any technical book Addison-Wesley had ever sold at a conference. The book is still doing very well. In fact, JOOP (Journal of Object-Oriented Programming) named it (in their September 1995 issue) both the best object oriented book of 1995, and the best object oriented book of all time.

1998: In 1998, the Gang Of Four were awarded Dr Dobbs Journal 1998 Excellence in Programming Award. In March, Inaugural ChiliPLoP conference was held in Wickenburg, Arizona, hosted by AG Communication Systems in Phoenix, Arizona. ChiliPLoP is a conference for learning how to write patterns and pattern languages and is host to Hot Topics.

1999: Second annual ChiliPLoP conference was held in March.

*A Proto Pattern Language for Human-Computer Interaction***Appendix 4:**

Patterns in Human-Computer Interaction

Background
Abstracts

*A Proto Pattern Language for Human-Computer Interaction***Appendix 4:**

Patterns in Human-Computer Interaction

Background

As I have mentioned in the previous section of this chapter, the first software patterns were WindowPerTask, FewPanels, StandardPanels, NounsAndVerbs, and ShortMenus. Ward Cunningham and Kent Beck wrote these patterns during consulting with "Tektronix's Semiconductor Test Systems Group." As their names show, main concerns of all of these five patterns were human-computer interaction.¹⁴⁰

Human-computer interaction's topics have been the subject of many patterns. Some of these patterns concentrate on the technical side and others are concentrated on the human side of the relation between human and computer.

The list of the published patterns at Plop and EuroPlop that are related to the field of human-computer interaction is included in Table 9.

Year	Conference	Pattern
1996	PloP [Plop, 96]	1 - "Storyboard Generator" an Architectural Pattern for Multimedia Applications 2 - Match/Mismatch 3 - Metaphor - A Pattern Language for User Centered Software Design 4 - Toward a Pattern Language for Hypermedia Applications
	EuroPlop	0
1997	PloP [Plop, 97]	1 - Pattern Systems for Hypermedia
	EuroPlop [EuroPlop, 97]	1 - Requested Interface 2 - Form-Based User Interface-The Architectural Patterns 3 - GOF Patterns for GUI Design
1998	PLoP [Plop, 98]	1- User Interface Software 2 - Interaction Patterns 3 - Patterns for Interactive Applications 4 - Patterns for Dynamic Websites 5 - Composing Multimedia Artifacts for Reuse
	EuroPlop [EuroPlop, 98]	0
1999	PloP [Plop, 99]	1- Usability Patterns for Applications on the World Wide Web
	EuroPlop [EuroPlop, 99]	1- A Pattern Language for Developing Graphical User Interfaces

Table 9: *List of the published patterns by Plop and EuroPlop that are related to the field of human-computer interaction.*

Abstracts

This part includes abstracts of the accepted patterns by Plop and EuroPlop that are related to the field of human-computer interaction. The variety of issues is remarkable.

Plop 96

1- "Storyboard Generator" an Architectural Pattern for Multimedia Applications.

Chrystalla C. Alexandrou and George A. Papadopoulos .

Multimedia systems are innovative systems that initially were meant to be used primarily for educational and presentation applications. The wide use of Internet and generally the "media war" that characterize our age, forces the use of multimedia in a lot of other types of applications. Innovative systems require innovative techniques and methodologies. This paper presents an architectural pattern for the creation of a StoryBoard. The StoryBoard is a technique that is borrowed from film making, and enhanced with additional characteristics from multimedia systems such as interactivity. This pattern assists the designer in generating a StoryBoard for the application that she is going to develop.

2- Match/Mismatch

Mark Bradac Becky Fletcher

This pattern describes a solution to the problem of managing a complex hierarchy of GUI interfaces where multiple sub canvases are selected based on state data. The pattern example and code fragments come from a synthesis application written in Smalltalk.

3- Metaphor - A Pattern Language for User Centered Software Design.

Todd Coram and Jim Lee.

This paper is introduced as the beginnings of a pattern language that can be used to generate software designs which are "user centered." Based on this paper user centered software designs place the user first and foremost. The authors were concentrating their patterns at the point where a user interacts with the application.

They claimed that by using these patterns, one should be able to develop languages which help to build user interfaces and structure a software to be driven by user interactions. There was no information on how to use icons, pop-up menus, dialog boxes and other interface objects but it could show "when to use them". The primary focus was on the higher-level patterns that can be found in all good user interfaces but it takes us down to a low enough level to give practical patterns which can be used in selection of tools.

4- Towards a Pattern Language for Hypermedia Applications

G. Rossi, D. Schwabe and A. Garrido

This paper presents two design patterns for the hypermedia domain including: "Navigational Contexts" and "Information by Demand." They are applied in two different aspects of hypermedia applications design: the design of healthy

navigational structures and the design of understandable and usable hypermedia interfaces, respectively. These two patterns are part of an effort for developing a pattern languages for the domain of hypermedia.

Plop 97

1- Pattern Systems for Hypermedia

Alejandra Garrido, Gustavo Rossi and Daniel Schwabe.

The hypermedia domain is currently receiving much attention, mostly due to the new generation of open systems, those that allow the connection among applications in the same or different machines (intranets) or those that publish the interface of an application in a WWW's browser. However, current applications in this domain are not taking profit of all benefits that characterize hypermedia applications, and maintenance is very difficult to achieve. Building large hypermedia applications is a hard task and although there exist many hypermedia design methodologies we also need design patterns that convey the expertise in the domain of hypermedia. This paper presents three pattern systems intended to provide guidance for different aspects of hypermedia applications. The first one concerns the development of software support for hypermedia in the context of object-oriented applications. The second one deals with organizing the navigational structures in a clear and meaningful way for intended readers and

the third one gives some guidelines for building an effective interface.

EuroPlop 97

1-Requested Interface

Ullrich Köthe

This paper introduces the Requested Interface pattern, which describes ways to implement truly independent software components that can be plugged together as needed in order to make reuse more attractive than re-implementation. It encourages components to delegate sub tasks to collaborating servers so that they can be adapted to a new context by simply exchanging those sub task servers. The delegating objects must specify minimal and abstract requested interfaces that describe the sub tasks independently of existing server interfaces. An adaptation layer mediates between the requested interface of a client and the offered interface of a server implementing the sub task.

2- Form-Based User Interface--The Architectural Patterns

Jens Coldewey and Ingolf Krüger.

Despite all of the benefits of object-oriented user interfaces, there are still domains that call for a form-based user interface. Business information systems that support fast processing of few, well-defined use cases are typical examples. This pattern language helps to develop the software architecture for such systems.

3-GOF Patterns for GUI Design

James Noble

The Design Patterns book introduced twenty-three patterns for object-oriented software design. These patterns are used widely but only in their intended domain of software design. We describe how six of these patterns can be used for the conceptual design of graphical user interfaces. By using these patterns, designers can produce interfaces which are more consistent, make good use of screen space, and are easier to use.

Plopp 98

1- User Interface Software

Jens Coldewey

This pattern language digs step by step into the design of user interface architecture. It does not go to the layout of a user interface. The paper considers this issue as a matter of ergonomics and bears enough potential to form a complete set of pattern languages on its own. Instead, this paper is about the software that drives the user interface. Figure 35 shows the overall landscape of the pattern language. It starts with the most fundamental pattern, the User Interface Layer.

Two patterns describe the architecture of this layer: **Separate Transformation**

**Figure 35: The overall landscape of the pattern language
(User Interface Software Pattern)**

explains how to deal with complex interactions while the Widget Model helps to structure presentation. Though both patterns seem to describe different philosophies, they are often combined to form the basic architecture of the User Interface Layer. Still, we need to deal with more issues. At first we have to provide Context Support between different interactions of the user. Depending on the requirements and architecture of our system we can apply several patterns for this. For the sake of brevity this document contains only thumbnails of them. Another area of interest concerns Domain Layer Access, which again lays the foundation of several other patterns. While some of them are well-known design patterns, others are special to user interfaces.

2- Interaction Patterns

Jenifer Tidwell

The fifty patterns contained in this language address the general problem of how to design a complex and interactive artifact. The language's goal is to support high-quality interaction between that artifact and its users. The patterns range from large-scale "genre" patterns, such as Form and Composed Command, to smaller special-purpose patterns like Reality Check and Short Description. The language should be of particular interest to software designers, although the patterns seek to capture design patterns across many different media, including print and industrial design. The language does not address implementation.

3- Patterns for Interactive Applications.

William C. Wake.

Many applications that we think of as "killer applications" on personal computers, such as word processors or spreadsheets, developed before graphical user interfaces were common. While graphical interfaces have brought several benefits to these programs (better usability, scrolling, fancy fonts, and crisper graphics), the core models of these programs haven't changed: word processors still deal with text, and spreadsheets with matrices of formulas.

This pattern language attempts to generalize the achievements of some applications such as Emacs, WordStar, dBASE III, and VisiCalc, and to capture their essence. These patterns apply fairly early in the design, after task analysis and object-oriented analysis have identified key tasks, objects, and relationships.

The patterns apply to interactive programs, be they terminal-based or graphical. By "interactive," it is trying to get at the difference between the *ed* and *vi* editors. The line editor *Ed* works nicely on a roll of paper and on a hard-copy terminal when working one line at a time, and the file is printed only at request. The *VI* editor works on a screen and it continuously shows the current document. We can enter text and see it in context as we type.

In the Portland Pattern Repository, Kent Beck has articulated two related patterns. In the *Story*, he notes that while stories are presented in time, the interface must be designed in space. In *WindowPerTask*, he provides guidance on how to associate windows with tasks. These patterns are intended to weave into that the same design space, and assume Beck's patterns as a given.

This pattern language addresses three aspects of application design: appearance, behavior, and extensibility as follow:

Structuring Metaphor discusses how to design screens,

Computer Magic discusses the important characteristics of application behavior,

Two-Layer Architecture discusses how to make an application extensible.

4- Patterns for Dynamic Websites

Fernando Lyardet, Gustavo Rossi .

This paper presents two hypermedia patterns for web sites to explore solutions that have been adopted to face common problems about the information available on the user and the web site sides. One aspect of this problem is analyzed on the News pattern to handle the problem of what do the users know about the changes on a web site. Another aspect is analyzed in the Dynamic Configuration pattern from the web site perspective, regarding the activities that are carried out by users and how these activities can be validated.

5- Composing Multimedia Artifacts for Reuse.

Jacob L. Cybulski, Tanya Linden.

This paper describes a pattern language that is used to define a multimedia-authoring environment capable of producing and utilizing multimedia components. It believes that the effective construction of multimedia material should refrain from the common practice of building new documents and presentations entirely from scratch. Hence, the proposed pattern language emphasizes the process of making new multimedia artifacts by reusing the existing components to suit the requirements of new applications. Each of the presented patterns describes one

well-known approach to multimedia authoring, e.g., joining and breaking artifact groups, defining and filling in templates, arranging and re-arranging artifact collections, creating and holding presentations, synchronizing multiple multimedia channels, etc. The paper also provides some insight as to the direction of the pattern language development, i.e., it lists six dimensions of multimedia authoring and reuse and shows what areas of functionality the final form of the pattern language must ultimately address.

Plot 99

1- Usability Patterns for Applications on the World Wide Web.

Kimberly Perzel, David Kane.

The World Wide Web (WWW) has dramatically changed the look and feel of GUI interfaces, sometimes for the better, sometimes to the user's detriment. There are solutions to the usability problems introduced by the web. This paper presents an initial collection of these patterns. These pattern and others like them endeavor to establish helpful guidelines for developers to use and eliminate the mystery for them and for the end user. These patterns document the human factors implications of frequently used interface elements for applications deployed over the web. Web applications require a much broader range of user input than web sites. Web application development often involves the collaboration of skilled information technology professionals with widely varying

backgrounds and terminology. Usability is a critical success factor for successful web applications. We believe that patterns can provide a language for communicating usability concerns among these diverse participants in the development process without hindering the creative elements of web development. The patterns included in this paper are: "Required Fields," "Carrot on a Stick," "What they see is all they get," "Policy Statement," and "Plan B."

Euro Plop 99

1- A Pattern Language for Developing Graphical User Interfaces.

Dwight Deugo and Dorin Sandu.

One goal for graphical user interface (GUI) development is that application Windows are easy to build, modify and maintain. An obvious way to meet this Goal is to have all developers use the same approach. Patterns and pattern languages are excellent vehicles for expressing common design and architectural approaches. The Observer pattern, the Model-View-Controller pattern and the pattern language for developing form style windows provide a partial set of patterns for building GUIs. After building GUIs for years, we found that more details were required for properly handling the event notification from the visual components and for updating them. We have incorporated these decisions into new patterns without changing the above-mentioned patterns, forming an ever-growing pattern language for GUI development.

*A Proto Pattern Language for Human-Computer Interaction***Appendix 5:**

Related Patterns

and Technological Concepts**Section 1: Organization and Complexity****Section 2: Frameworks and Pattern****Section 3: State and its Patterns****Section 4: Window and its Patterns****Section 5: Panel and its Patterns****Section 6: Widgets and Patterns****Section 7: Joint Application Development and Participatory Design**

*A Proto Pattern Language for Human-Computer Interaction***Appendix5: Section 1:****Organization and Complexity**

A Proto Pattern Language for Human-Computer Interaction

Appendix5: Section 1:

Organization and Complexity

This part does not introduce a pattern but is a support document for the Scalability Pattern in my proto-pattern language. ¹⁴¹

Organization and Complexity

Complexity theory is an emerging field of study in science and technology. Fundamentally, a system is complex when it is beyond analysis by the standard methods of systems analysis. Complex systems are ones in which patterns can be seen and understood but the interplay of individual elements cannot be reduced to the study of them.

There are a number of "sciences of complexity," each dealing with approaches to understanding the dynamic behavior of units that range from organisms to the largest economic, social, and political organizations. Often, such studies involve complex adaptive systems and hierarchical systems. Complexity studies are generally multi-disciplinary in nature. They attempt to pursue knowledge and discovery of the features that are shared by all the different systems that are described as, somehow, "complex." These studies are known by such different names as complex adaptive systems, complex systems theory,

complexity theory, dynamical systems theory, complex nonlinear systems, and computational intelligence. Sometimes, the phrase "chaos theory" is used, but chaos has a popular connotation that is not necessarily implied by this phrase.

Systems can be modeled in the sense that one can describe an analogous system that, in some ways, replicates the behavior of the original system for a given class of inputs. Many scientific studies, prior to the development of simulation models and complexity theory, involved trying to understand the world through the use of linear models. When a study resulted in anomalous behavior, the failure was often blamed on noise or experimental error. Now, it is recognized that this noise or experimental error provides important information. It was the study of this noise, then called chaos, that led to the initial use of the term "chaos theory" to describe what is now called "complex systems theory."

Complexity of a system may be determined by the complexity of the model that is necessary to predict the behavior of a system. The more the simulation model resembles the actual system and yields the same behavior, the more complex the system is said to be. With a complex nonlinear system, very small changes to the initial conditions of the system's states, or inputs to the system, will result in very different outputs for that system. A general rule for complex systems is that we cannot create a model that will accurately predict the outcomes of the actual system. However, we can create a model that will

accurately simulate the processes that the system will use in order to create a given output. This awareness has profound impacts for organizational efforts. It also calls forth many concerns relating to the real value of creating organizational plans and mission statements and suggests instead the creation of a model of the planning process itself and subjecting this to various inputs in order to generate output scenarios.

The Complexity theory has developed from four major knowledge areas including:

1. Mathematics, through the use of various approaches, such as fractals, cellular automata, and other nonlinear, graphical mathematical models.
2. Modeling, which enables computer simulation of the dynamic behavior of systems.
3. Biology, through the identification of new evolutionary processes and genetic algorithms that lead to understanding genetics, artificial life simulations, and cognitive behavior.
4. Physics, especially thermodynamics and the entropy concept, which enables the study of turbulence and understanding of self-organizing systems and such system states as equilibrium, near equilibrium, edge of chaos, and chaos.

Teams of biologists, economists, engineers, and organizational specialists have undertaken integrated studies of complexity theory and organizational knowledge. Two of the major conclusions of these studies are including:

- 1. The potentially most productive use of complexity theory is in the generation of new possibilities for technological innovation, and that these possibilities require new ways of thinking and new models.**
- 2. The organizational knowledge and results are a strong function of the ways in which organizational processes result in action and learning.**

These studies of complex systems often run completely counter to the trend toward increasing fragmentation and specialization in most disciplinary specialties. They have stimulated interest in a broad synthesis of knowledge involving mathematics, computational sciences, physics, chemistry, biology, neuroscience, social science, economics, political science, and engineering. The current trend in complexity studies is to reintegrate the fragmented interests of most disciplinary specialties into a common pathway leading to the understanding of the nature of many, complex real-world systems, and to develop our capability to regulate or control such systems in the interest of life support and sustainable development.

*A Proto Pattern Language for Human-Computer Interaction***Appendix5: Section 2:**

Frameworks and Patterns

What is a FrameWork?

The advantages of using frameworks

Frameworks and Class Libraries

Frameworks and Patterns

*A Proto Pattern Language for Human-Computer Interaction***Appendix5: Section 2:**

Frameworks and Patterns

The concept of a framework and the concept of a pattern are very similar. To distinguish these two concepts from each other I look briefly to the concept of a framework in this section

What is a framework?

A framework is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions.

Taligent (the former IBM subsidiary) has designed frameworks for system software functions, such as networking, multimedia and database access.

[Taligent, 93] ¹⁴²

With frameworks, software developers don't have to start from scratch each time they write an application. Frameworks are built from a collection of objects, so both the design and code of a framework may be reused.

The process for developing frameworks has four important guidelines:

- 1- Derive frameworks from existing problems and solutions.
- 2- Develop small, focused frameworks.
- 3- Build frameworks using an iterative process driven by client participation

and prototyping.

4- Treat frameworks as products by providing documentation and support, and by planning for distribution and maintenance.

A framework captures the programming expertise necessary to solve a particular class of problems. Programmers purchase or reuse frameworks to obtain such problem-solving expertise without having to develop it independently. Also, a framework helps developers provide solutions for problem domains and better maintain those solutions. It provides a well-designed and thought out infrastructure so that when new pieces are created, they can be substituted with minimal impact on the other pieces in the framework. [Nelson, 94] As you can see the idea of the framework is very similar to the pattern.

The Advantages of using Frameworks

Frameworks are a long-term investment. The benefits gained from developing frameworks are not necessarily immediate. The key to maximizing the benefits of writing frameworks is to get as many developers as possible using them. The disadvantages of using frameworks are as follows:

1- Framework designers need more time to create a framework than a procedural library,

2 - Clients need more time to learn a framework than a procedural library,

3- A framework must be supported by the team's processes and

organization.

Figure 36 shows the balance between costs and benefits of using a framework.

Figure 36: The costs and benefits of using a framework.

Frameworks and Class Libraries

While clear differences exist between class libraries and frameworks, some libraries exhibit framework-like behavior, and some frameworks can be used like class libraries. You can view this as a continuum, with traditional class libraries at one end and sophisticated frameworks at the other.

Figure 37 illustrates the differences between class libraries and frameworks.

Figure 37: Framework and Class Library

Patterns and Frameworks

As you begin to design how the framework will work, you might also discover that you can break the framework down into a collection of recurring design patterns, much the way you decompose the initial problem into a set of frameworks. Each design pattern is a micro-architecture for a recurring element. Patterns can represent generic software elements, or elements that are particular to a problem domain. When you design a framework, look for recurring patterns that can be applied to other problems.

Figure 38 illustrates the relationships between Framework and Design Patterns.

Figure 38: Framework and Design patterns.

Appendix5: Section 3:

State and its Patterns

State Pattern (Change an Object's Class at Run-Time)**Example**

Appendix5: Section 3:

State and its Patterns

State Pattern (Change an Object's Class at Run-Time)

The State Pattern is a solution to two problems: how can you change the class of an object at run-time? And how can you model a finite state machine using object-oriented techniques? In other words, a monolithic object's behavior is a function of its state, and it must change its behavior at run-time depending on that state. Or, an application is characterized by large and numerous case statements that vector flow of control based on the state of the application. The State Pattern allows an object to alter its behavior when its internal state changes. The object will appear to change its class. Therefore, the State pattern is a solution to the problem of how to make behavior depend on state. ¹⁴³

To apply the State Pattern you should follow these steps:

- 1- Define a "context" class to present a single interface to the outside world.
- 2- Define a State abstract base class.
- 3- Represent the different "states" of the state machine as derived classes of the State base class.

- 4- Define state-specific behavior in the appropriate State derived classes.**
- 5- Maintain a pointer to the current "state" in the "context" class.**
- 6- To change the state of the state machine, change the current "state" pointer.**

The State Pattern does not specify where the state transitions will be defined. The choices are two: the "context" object, or each individual State derived class. The advantage of the latter option is ease of adding new State derived classes. The disadvantage is each State derived class has knowledge of (coupling to) its sibling which introduces dependencies between subclasses. [Gamma, et al., 95]

A table-driven approach to designing finite state machines does a good job of specifying state transitions but it is difficult to add actions to accompany the state transitions. The pattern-based approach uses code (instead of data structures) to specify state transitions but it does a good job of accommodating state transition actions. [Gamma, et al., 95]

The implementation of the State Pattern builds on the Strategy pattern. The difference between State and Strategy is in their intents. With Strategy, the choice of an algorithm is fairly stable. With State, a change in the state of the "context" object causes it to select from its "palette" of Strategy objects. [Coplien, 99-3]

This pattern can be observed in a vending machine. Vending machines have

states based on the inventory, amount of currency deposited, the ability to make change, the item selected. When currency is deposited and a selection is made, a vending machine will either deliver a product and no change, deliver a product and change, deliver no product due to insufficient currency on deposit, or deliver no product due to inventory depletion. [Duell, 97]

Example

Referring to Perkins's example of the work space for editing an address entry in the email handler can be helpful in having a general view of the State Pattern. [Perkins, 97] He explains that this work space can be represented as is shown in Figure 39.

Figure 39: A Workspace for editing an address entry in the email handler.

In this example, Perkins is modeling the differing meanings of edit, save and

quit depending on the state of the sub-application. When implementing such a design, one can use a "state" attribute to track an object's state and write the appropriate conditional code in her state-dependent operations.

According to Perkins, this may be good enough for simple models. But with larger models it becomes very clumsy. The actual processing becomes hard to read and debug because it is deep inside conditionals, and is mixed up with the state transition handling. Changes to the model cause huge changes to the code of every method. The object with state behavior is split into a Context and a State. The context object contains the common elements of the object's state, and dispatches all state-dependent commands to its state attribute. The state object contains state-specific attributes and implementations for each state-dependent command. A separate class is defined for each state. Figure 40 represents a *Workspace* object when it is in the *Saved* state.

In this schema, *Workspace* is the context that is linked to an object representing its current state. Therefore, *Workspace* methods simply dispatch to the current state. Methods in the state do not need condition analysis and have no concern for processing in other states. ¹⁴⁴

In my thesis, the idea of changing the state of main nodes of an informative artifact according to the different situations with which a user views that node, is a representation of change of states. This idea is inspired by the State Pattern

and represents the same technological procedure in a meaningful appearance to the user.

Figure 40: Workspace object when it is in the Saved state.

*A Proto Pattern Language for Human-Computer Interaction***Appendix5: Section 4:**

Window and its Patterns

Definitions**Window where it starts****TaskWindow Pattern****Technological Background** **Model-View-Controller****Model-View-Controller as an Aggregate Design Pattern****Model-View-Controller with Presentation Model****Visual Proxy Pattern**

*A Proto Pattern Language for Human-Computer Interaction***Appendix5: Section 4:**

Window and its Patterns

Definitions:

In Almanac's Computer Glossary window is defined as: " A portion of a computer display used in a graphical interface that enables users to select commands by pointing to illustrations or symbols with a mouse." [Almanac, 99]

Java considers a window as a top-level window, with no borders and no menu bar. The AWED sends the window all mouse, keyboard, and focus events that occur over it. The window behaves as a modal dialog in that it will block input to other application windows when showed. Each window needs the main application frame as the parameter to be constructed. It will be opened in that frame. [Gosling, et al., 96-1]

My thesis considers "window" as the abstract representation of the content or parts of the content in the interaction domain. Window is where that space of an informative artifact is represented. It is an analogue to "Shape" for "Space" in architectural space, and "Window" for main application frame in Java.

Window, where it starts:

Ward Cunningham and Kent Beck during consultation with "Tektronix's Semiconductor Test Systems Group" in 1988, decided to try out the pattern to solve some problems in design. Using Alexander's ideas, Ward and Kent let representatives of the users (a trainer and a field engineer) finish the design. They came up with a five-pattern "language" that helped the novice designers take advantage of Smalltalk's strengths and avoid its weaknesses:

WindowPerTask, FewPanels, StandardPanels, NounsAndVerbs, ShortMenus.

WindowPerTask was rewritten by Kent Beck for Smalltalk Best Practice Patterns later as: *Task Window*. This pattern effectively sides steps gross problems *that Model - View - Controller* once had with mutually dependent windows. Cunningham and Beck were amazed at the elegance of the interface their users designed. They reported the results of this experiment at OOPSLA 87 in Orlando. They wrote up a panel position and presented at Norm Kerth's workshop on "*Where do objects come from?*" Historically these patterns are considered the start point of pattern writing and later the pattern movement in the software domain. [Cunningham, 99]

Task Window Pattern (Window Per Task)

You have found a Task.

How much of the system do you reflect in a single window? Interface design

tends to fall into one of two camps: the "everything in one window " camp and the "everything in separate windows" camp. Neither of these extremes serves the user of current computer systems.

The "everything in separate windows" is exemplified by early design on the Macintosh. With the extreme limits it had on screen real estate, it was appropriate not to pack too much into any window. However, it resulted in some software that was hard to use (Finder, ResEdit) because the user spends far too much time managing windows. The need for a separate "Window" pull down menu is a symptom of this problem.

"Everything in the same window" goes too far the other way. Putting all the functionality of a large system (many workstation CAD systems demonstrate this problem) into one window results in a vast sea of choices. The system isn't helping the user structure their time and isn't presenting any flow through the system. Finding a happy medium is critical to the operation of the interface (and to the viability of the rest of these patterns). How do you tell when you have enough information in a window?

Ask the user to divide their job into identifiable tasks. Create a window for each task. Name the window after the task.

The available actions will be *Visible Functions*. Divide the window into a Few Panes. ¹⁴⁶

Technological Background

Model-View-Controller

Model-View-Controller is the concept introduced by Smalltalk's inventors¹⁴⁶ of encapsulating some data together with its processing (the model) and isolating it from the manipulation (the controller) and presentation (the view) part that has to be done on a user interface.¹⁴⁷

1. A model therefore is an object representing data or even activity, e.g., a database table or even some plant-floor production-machine process.
2. A view is some form of visualization of the state of the model.
3. A controller offers facilities to change the state of the model.

Figure 41 is a representation of the Model-View-Controller relationship.

Figure 41: Model-View-Controller relationship.

Model-View-Controller as an Aggregate Design Pattern

According to Ralph Johnson Model View Controller is made up of a number of

smaller design patterns. The views form a tree using the Composite Pattern. The relationship between views and models is the Observer Pattern. The controllers are strategies of the views. Document View is more popular now than true Model View Controller and it consist of Composite and Observer, but not Strategy.

[Gamma, et al., 95] Although Composite, Observer and Strategy are the core of Model View Controller, it uses more patterns. The model is often a Mediator (Example: Smalltalk Browser). The tree of views often contains Decorators to add properties to views. A view often uses an Adaptor to convert the model to a standard interface that the view can use. Views create controllers using a Factory Method.

Model-View-Controller with Presentation Model

A good comparison and explanation about Document View and Model-View-Controller can be found in the stingray.com home page. [Stingray, 99] This document explains why the model should be divided to System Model and Presentation Model. Then, there will be four classes:

1. MvcModel-based System Model - Problem Domain Data.
2. MvcPresentationModel_T - Presentation Domain Data.
3. MvcViewport - Perspective Domain Data.
4. MvcController - Selection State.

The MVC Class Hierarchy is shown in Figure 42.

Figure 42: The MVC Class Hierarchy.

The presentation model creates and maintains `MvcVisualComponents` that reference a node or object in the system model. The presentation model also receives notifications of change from the system model. The presentation model should act upon the notifications it receives from the problem domain by creating, destroying or invalidating visual components. As a result, the view port receives a change notification or invalidation request from the presentation model. Figure 43 shows the relationships between the models in a Model-View-Controller with Presentation Model.

Figure 43: Model-View-Controller with Presentation Model.

Example: Assume the system model models a nuclear reactor. If a dangerous condition arises, the monitoring system would provide input to the system model. The system model would set a flag internally and forward a notification of the change to all observers. The presentation model would receive the notification and create a `MvcVisualComponent` object to present a red, flashing message. The presentation model would broadcast an invalidation request to all renders, which would cause the viewports to paint the alert.

Visual Proxy Pattern

The other solution for minimizing the coupling relationships between the GUI subsystem and the underlying logical model is "Visual Proxy Pattern." [Holub, 99] The Visual-Proxy Pattern is in some way a specialization of the Presentation / Abstraction / Control (PAC) architecture described in *Pattern Oriented Software Architecture: A System of Patterns*, by Buschmann and others, [Buschmann, et al.,

96] as shown in Figure 44. In this pattern a three-layer architecture is used. These three layers include:

1. The abstraction layer

The abstraction layer in the PAC architecture or logical model is the group of objects that directly models abstract concepts in the problem domain.

The classes that define these objects are the product of the analysis process, and shouldn't have a specific presentation associated with them.

The abstraction layer is a rough equivalent to the "model" in MVC.

2. The presentation layer

Those classes that are responsible for drawing on the screen comprise the presentation layer. These classes render one or more attributes of the classes in the abstraction layer. They are visual proxies because they really are proxies for representatives of abstraction-layer objects. The control layer requests visual proxies from the underlying abstraction layer. These proxies are quite abstract from the perspective of the control object, so essentially there is no coupling between the control layer and the other two layers.

3. The control layer

The control object's job is to build a view by requesting proxies from the model side and installing them in a window. The control object, then, is in

charge of layout and large changes to the user interface. A control object can itself be an abstraction-layer object, whose presentation is an entire form. This recursive architecture, is what the Gang of Four refers to as the Composite pattern. [Gamma, et al., 95]

Figure 44: The Visual Proxy and its Layers.

Though the abstraction and presentation layers are loosely analogous to the model and view of MVC, it has no notion of a control object. A MVC "controller" is an object that accepts events from the view side and translates them into messages on the model side. A PAC control object, on the other hand, is passive with respect to message flow. The messages go directly from the visual proxy to the model-side object that manufactured the proxy. The window on which the view is built (which is typically the user interface of the control object itself) usually is not involved in this message flow.

*A Proto Pattern Language for Human-Computer Interaction***Appendix5: Section 5:**

Panel and Patterns

Definitions**FewPanels Pattern**

A Proto Pattern Language for Human-Computer Interaction

Appendix5: Section 5:

Panel and Patterns

Definitions

Java considers Panel as the simplest container class that provides space into which an application can attach any other component, including other panels.

The AWT sends the panel all mouse, keyboard, and focus events that occur over it. The panel has a default layout manager, which is called FlowLayout layout manager. [Gosling, et al., 96-2]

My thesis considers "panel" as the visual representation of a part of the content in the interaction domain. A panel delimits a window to reach the visual representation. It is an analogue to "surface" for "shape" in architectural space, and definition of panel in Java.

The background of the FewPanels Pattern as one of the related patterns for the WindowPerTask Pattern is explained in previous part of this section. This pattern is an important and subtle patterns that in spite of its obvious benefits, there are some misunderstanding about that.

As an example look at the set of "Visualesson Patterns" by Ian Chai. [Chai,

2000] He rewrites the WindowPerTask Pattern. This set of patterns omits the panel as a mediator between a window and widgets. Therefore, in his set of patterns the abstract concept of the window relates directly to the concrete widgets.

By looking at this set of patterns, I conclude that one should have a good reason to rewrite a pattern. Therefore, the modification to a pattern should respect the logic of the original one, carefully.

Few Panes Pattern

To understand complex things one often needs to see it from several points of view.

Therefore: Provide these points of view (called simply Views) by dividing the area of your WindowPerTask into panes.¹⁴⁸

*A Proto Pattern Language for Human-Computer Interaction***Appendix5: Section 6:**

Widgets and Patterns

A Pattern Language for Human Computer Interaction Design
Iconic Reference Pattern

A Proto Pattern Language for Human-Computer Interaction

Appendix5: Section 6:

Widgets and Patterns

A Pattern Language for Human-Computer Interaction Design

Tidwell introduced her pattern language for Human-Computer Interaction Design to Plop98. The date of the last modification of this language is 17 May 1999. ¹⁴⁸

Tidwell's pattern language includes 60 patterns that are divided into ten groups. According to her, these patterns are intended to form an Alexandrian pattern language, as found in Christopher Alexander's book "A Pattern Language," [Alexander, et al., 77] and not a catalog such as is found in the book "Design Patterns." [Gamma, et al., 95] This means that they are intended to be used together synergistically, in such a way that the whole is more than the sum of its parts. This pattern language does not break new theoretical ground or present innovative new techniques. Instead, it captures ordinary design wisdom in a practical way. Tidwell did not attempt to address implementation issues. Seven of the patterns are not written. ¹⁴⁹ She did not draw a coherent diagram of the whole language and not any clear linear path between the patterns is defined.

Tidwell introduced the concept of sub-languages in her pattern language. According to her, each primary pattern tends to use certain other patterns more than the others. Therefore, she loosely grouped these patterns together as sub languages.

The target audiences of Tidwell's pattern language are the people who design, implement, or test the use-ability of traditional user interfaces, Web sites, on-line documentation, video games, and other such things, as well as the people who manage the teams to do so.

Through her pattern language Tidwell tried to find solutions for following two problems: "How to design a complex interactive software artifact," and "How to support high quality interaction between a person and a software artifact."

The artifact may support one or more of a broad spectrum of activities, ranging from the most passive (absorbing information with little or no interactivity) to the hands-on creation of other objects. Examples of the products that Tidwell refers to throughout her pattern language include: Traditional desktop GUI software, Web sites, Palmtops and handhelds, Controllers for industrial processes, Video games, On-line documentation, Multimedia educational software, Speech-based interfaces, Design tools for Multimedia graphics and visual arts, Scientific visualization, Collaborative and multi-person workplaces.

These products are diverse in their specific goals, their degrees of interactivity, their balance between the verbal and the nonverbal, and other

factors. But from a designer's perspective, there are commonalities among them. For one thing, many of them bear strong similarities to the older, more traditional media. Software is very young, and software designers are still learning how to take full advantage of its unique characteristics. In the meantime, both users and designers carry over what we already understand about print media (books, paper forms, charts), mechanical things (appliances, cars), physical spaces (architecture, urban design), and other real-world artifacts. Therefore, many patterns in this language draw as much from these other realms as from software.

Another similarity among all these kinds of software (and other media) is their basic purpose. The best ones all provide their users with a successful and satisfying experience, by doing one or both of these things well:

- 1- They shape the user's understanding of something, through a stylized presentation that unfolds the content to the user in an appropriate way. A successful artifact will enable its users to completely understand and effectively use the content being presented.

- 2- They enable a user to accomplish a task, by progressively unfolding the action possibilities to the user at an appropriate pace as the user interacts with it. A successful artifact will "flow" so well that it lets its users focus entirely upon the task at hand, causing the artifact itself to fade from the

user's awareness.

These twin goals, along with corollary issues such as learn-ability, user empowerment, and enjoy-ability define "high-quality interaction" for the purposes of Tidwell's pattern language. Some artifacts concentrate more upon one aspect than the other, depending upon their context of use. The two aspects are almost orthogonal, but not quite: something which chiefly provides a set of actions still has to present those actions in a comprehensible way, and something which chiefly presents a set of facts or ideas may have to provide the user with actions they need to interact with the artifact.

In both cases, there is an "unfolding" process going on between the artifact and the user. In something which presents facts or ideas (such as a map) that unfolding process may be top-down, in which the "big picture" is shown first and then the users work their way down into the details as they need. Or, it might be in the form of a fictional narrative in which the author uses language and character to let the central themes slowly unfold. Therefore, according to Tidwell, the basic shape of the content might take the form of one of these primary patterns: Narrative (such as works of fiction, nonfiction books, and news articles written in the "pyramid" form), High-density Information Display (such as Maps, tables, and charts), and Status Display (A wall clock, a car's dashboard, and a VCR display).

In an artifact which presents actions, such as a GUI "wizard," the user may first be presented with a small range of available actions, one of which is taken; then a new set of possible actions is shown, and the user takes one of those; and so on. Alternatively, the range of actions at any time might be very wide, as with some direct-manipulation interfaces. Ultimately, the artifact lets the user accomplish some principal task which may require smaller supporting tasks to be accomplished first. These later tasks may in turn require still smaller supporting tasks. The sense of "flow" comes from being able to do these with an appropriately small amount of time and effort, so that the user never loses her focus on their principal task.

In Tidwell's pattern language the primary patterns for actions might include these familiar genres: Form (Tax returns and catalog orders), Control Panel (a light switch), WYSIWYG Editor (a typewriter or word processor, a drawing program, and a forms designer such as Visual Basic), Composed Command (a Command-line-driven software of all kinds, such as the UNIX operating system or software debuggers), and Social Space (the on-line equivalents of "real" social spaces such as newsgroups, email lists, chat rooms).

These primary patterns form the backbone of Tidwell's pattern language. They set the tone for an artifact. When a user identifies an unfamiliar artifact as belonging to one of these, she tends to make some initial assumptions about its

behavior, based on cultural expectations of how these things usually work.

The patterns can be used in combination with each other, but the boundaries are basically respected by mainstream artifacts, and for good reason. Usability is improved when users' expectations, subconscious or explicit, are followed. On the other hand, experiments, cutting-edge interfaces, and art often make it a point to violate those boundaries. How a designer deals with the existing boundaries all depends upon your purpose.

The next set of patterns captures ways of unfolding an artifact's content or available actions. Some apply to content, some apply to actions, and some to both. **Navigable Spaces** pattern is highly interactive, letting users move through the artifact at their own paces, and in their own directions. **Overview Beside Detail** pattern also lets users work at their own paces but presents a more structured, two-level view of information or actions. **Step-by-Step Instructions** pattern prescribes a sharply limited set of actions to the users, usually in a linear fashion, moving progressively through stages. **Small Groups of Related Things** pattern loosely organizes visual content into distinct groups, usually done hierarchically, so that a user can see both the "big picture" and fine details. **Series of Small Multiples** pattern organizes visual content into a single long series of discrete images, bringing out differences between them by emphasizing their basic continuity. It can be used to implement a High-Density Information Display. **Hierarchical Set** pattern is a way of implementing a High-Density

Information Display with a strict treelike organization. **Tabular Set** pattern also implements High-Density Information Display, but as a table; again, users can see both the big picture and fine detail. **Chart or Graph** pattern is yet a third implementation of High-Density Information Display, but a more graphic one than the other two. **Optional Detail On Demand** pattern lets users reveal hidden actions or content at their own discretion. **Disabled Irrelevant Things** pattern blocks certain actions or content according to the current state of the artifact, usually driven by user interaction. **Pointer Shows Affordance** pattern temporarily reveals possible actions according to the user's explicit focus of attention. **Short Description** pattern temporarily reveals content according to the user's explicit focus of attention.

Other categories of patterns in Tidwell's pattern language describe an artifact's use of space and other resources, navigational techniques, different kinds of actions, the interrelationships between an artifact's working surfaces, and so on.

Most of these patterns can be used at many different levels of scale. A Form pattern, for example, may be the dominant pattern in one artifact, while being a minor helper task in another. Likewise for a High-density Information Display such as a chart or a table. **Small Groups of Related Things** pattern is recursive by definition, much like Composite pattern in Design Patterns, [Gamma, et al., 95] and the concept of a working surface is also recursive. Any single surface can be

composed of a set of Tiled Working Surfaces, for instance. According to Tidwell, because of this scale issue, she has not been able to draw a coherent diagram of the whole language, nor define some clear linear paths through it.

I think that her pattern language suffers from the lack of organizational or tasks oriented button-up approach in organizing its patterns inside its framework. She wrote down her detailed experiences (or field's experiences) first, and finally tried to set up a coherent diagram for those. How can she guarantee the possibility of drawing such a diagram? This approach is in contrast with her claim that "these patterns are intended to be used together synergistically, in a way such that the whole is more than the sum of its parts." To create such a whole, you need to apply a top-down approach (just like object-oriented design paradigms).

Another weakness of her work is the format of her pattern language. There are many patterns in her pattern language with the similar problem definitions, introducing different solutions for the same problem. If all of those are different solutions for different parts or special cases of the same problem, a pattern should introduce those parts or cases, and the others implement the different cases introduced in the main pattern. It means that they should define the solutions to each derived (specialized) parts or case of the problem. Therefore, the problem definitions of those derived patterns will not be the same. Alexander followed the same derived procedure when he wrote down Pattern Language.

[Alexander, et al., 77]

Otherwise, if different solutions for the same problem exist and she wanted to introduce those, she should implement only one pattern. She could have described different solutions and by comparing those verified how the different context or forces would have impacted on the decision of a designer to apply one of those solutions.

After reading all of her patterns, I found out that she pursued the first option. Her different patterns with the same problem definitions are the solutions for the different formations or parts of one problem. In fact, in the existing format of Tidwell's pattern language you cannot find your pattern of choice by reading the problem definitions of a pattern. Also, it is not possible to write down a Problem-Solution summary for pattern language, as she did not do.

Tidwell, by combining solution and rationale headings of patterns of her language, documented a long solution heading for each pattern that makes the readers confused and tired. Also her solutions are not concrete and looks like a descriptive and subjective essay.

Tidwell's pattern language in spite of its weaknesses is the most comprehensive set of patterns that are written for the human-computer interaction design. Because of the lack of an organizational perspective, although these patterns can be linked to each other, I cannot think of those as a pattern language. I can consider those as the patterns for the widgets that can be combined to create an artifact if you have an organizational or structured plan for that artifact.

Table 10- Problem/Solution Summary of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (part 1)

No	Name	Problem	Solution
1- What is the basic shape of the content?			
1	Narrative	In what form should the information be displayed to the user?	Convey the information via natural language.
2	High-density Information Display	In what form should the information be displayed to the user?	Pack as much information into one working surface as possible, following the precepts of good graphic design, with an organization that accurately reflects the underlying structure of the information.
3	Status Display	How can the artifact best show the state information to the user?	Choose well-designed displays for the information to be shown. Put them together in a way that emphasizes the important things, de-emphasizes the trivial, doesn't hide or obscure anything, and prevents confusing one piece of information with another.
2- What is the basic shape of the actions taken with the artifact?			
4	Form	How should the artifact indicate what kind of information should be supplied, and the extent of it?	Provide appropriate "blanks" to be filled in, which clearly and correctly indicate what information should be provided. Visually indicate those editable blanks consistently, such as with subtle changes in background color, so that a user can see at a glance what needs to be filled in. Label them with clear, short labels that use terminology familiar to the user; place the labels as close to the blanks as is reasonable. Arrange them all in an order that makes sense semantically, rather than simply grouping things by visual appearance

Table 10- Problem/Solution Summary of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (part 2)

No	Name	Problem	Solution
5	Control Panel	How can the artifact best present the actions that the user may take?	For each function or state variable that is part of the user's mental model, choose one well-designed control that performs the function or displays the variable's value; put them all together such that the most commonly-used controls are the most prominent.
6	WYSIWYG Editor	How can the artifact best present what is being created, and what the user needs to do to create or change it	Always show the user an accurate and up-to-date representation of the artifact they are creating; allow the user to interact directly with it as they add to it, delete from it, modify it, and so on.
7	Composed Command	How can the artifact best present the actions that the user may take?	Provide a way for the user to directly enter the command, such as by speech or by typing it in.
8	Social Space		
3- How does the content or available actions unfold before the user?			
9	Navigable Spaces	How can you present the content so that a user can explore it at their own pace, in a way which is comprehensible and engaging to her?	Create the illusion that the working surfaces are spaces, or places the user can "go" into and out of.
10	Overview Beside Detail	How can you present this large amount of content so that a user can explore it at their own pace, in a way which is comprehensible and engaging to her?	Show the whole set of objects, or the whole undetailed data set, in one part of the display area, to act as an overview of the content.

Table 10- Problem/Solution Summary of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (part 3)

No	Name	Problem	Solution
11	Step-by- Step Instructions	How can the artifact unfold the possible actions to the user in a way that does not overwhelm or confuse them but instead guides them to a successful task completion?	Walk the user through the task one step at a time, giving very clear instructions at each step.
12	Small Groups of Related Things	How should the items or actions be organized?	Group the closely-related things together, nesting them in a hierarchy of groups if needed.
13	Series of Small Multiples.		
14	Hierarchical Set	How should the information be organized?	Show the data in a tree-like structure.
15	Tabular Set	How should the information be organized?	Show the data in a table structure.
16	Chart or Graph	How should the information be organized?	Show the data plotted against time or some other variable. Plot it together with other variables for further comparison.
17	Optional Detail On Demand	When should these usually-unneeded items be presented to the user, and how?	Up front, show the user that which is most important and most likely to get used. Details and further options which won't be needed most of the time – say 20% or less of expected uses – can be hidden in a separate space or working surface.
18	Disabled Irrelevant Things	How can the artifact steer the user away from actions that cannot or should not be taken, while still maintaining visual calm and stability?	Disable the things which have become irrelevant.

Table 10- Problem/Solution Summary of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (part 4)

No	Name	Problem	Solution
19	Pointer Shows Affordance	How can the artifact indicate that a visual entity represents an action that the user may take?	Change the affordance of the thing as the pointer moves over it.
20	Short Description	How should the artifact present additional content, in the form of clarifying data or explanations of possible actions, to the users that need it?	Show a short description of a thing, in close spatial and/or temporal proximity to the thing itself.
4- How does the artifact generally use space and the user's attention?			
21	Sovereign Posture	How should this artifact relate spatially to other artifacts that might share its space, and how can it best use the space it has?	Allow the artifact to take up all the space it needs to get the job done efficiently and gracefully.
22	Helper Posture	How should this artifact relate spatially to other artifacts that might share its space and how can it best use the space it has?	Use as much space as needed to make it comprehensible, but no more; focus tightly on the activity by excluding all but the commonest actions, and information, from the top level, but let those common ones take whatever space they need.
23	Background Posture	How should this artifact relate spatially to other artifacts that might share its space and how can it best use the space it has?	Make the artifact small, relative to the other primary activities going on at the same time, and keep it unobtrusive.

Table 10- Problem/Solution Summary of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (Part 5)

No	Name	Problem	Solution
5- How is the content or action organized into working surfaces?			
24	Central Working Surface	How should the artifact's working surfaces be organized?	Create one working surface where the artifact's major functions are collected together; if most of the work can actually be done there, so much the better.
25	Tiled Working Surfaces	How should the artifact's working surfaces be organized?	Place the working surfaces together in a plane, such that they do not obscure each other, and show the whole thing to the user.
26	Stack of Working Surfaces	How should the artifact's working surfaces be organized?	Stack the surfaces together. Label each surface with a unique and recognizable name or icon (or let the user pick the label), and visually cluster those labels together near the stack.
27	Pile of Working Surfaces	How should the artifact's working surfaces be organized?	Stack the surfaces loosely so that they obscure each other most of the time, but so that one or more surfaces of the user's choosing can be on top.
6- How can the user navigate through the artifact?			
28	Map of Navigable Spaces	How can the artifact help a user navigate effectively and remain oriented?	Provide a map or diagram of the Navigable Spaces relevant to the artifact.
29	Clear Entry Points	How does the user know where to start?	Provide a small set of well-defined, clearly-named entry points to the network of Navigable Spaces .
30	Color-Coded Sections	How can an artifact both give users a sense of place, and also tell them where they are, within a large network of spaces?	Use color to identify the major sections of the artifact.

Table 10- Problem/Solution Summary of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (part 6)

No	Name	Problem	Solution
31	Go Back One Step	How can the artifact. make navigation easy, convenient, and psychologically safe for the user?	Provide a way to step backwards to the previous space or state.
32	Go Back to a Safe Place	How can the artifact make navigation easy, convenient, and psychologically safe for the user?	Provide a way to go back to a checkpoint of the user's choice.
33	Convenient Environment Actions	How should the artifact present these actions?	Group these actions together, label them with words or pictures whose meanings are unmistakable and put them where the user can easily find them regardless of the current state of the artifact.
34	Localized Object Actions	How should artifact present the actions that may be taken on those objects?	Group object actions together, even more so than for Convenient Environment Actions , and spatially localize them to the object.
35	Actions for Multiple Objects	How can the artifact make repetitive tasks easier for the user?	Allow the action to be performed "in parallel" across a set of user- selected objects.
36	Choice from a Small Set	How should the artifact indicate what kind of information should be supplied?	Show all the possible choices up front, show clearly which choice(s) have been made, and indicate unequivocally whether one or several values can be chosen.
37	Choice from a Large Set	How should the artifact indicate what kind of information should be supplied?	Clearly show the selected value up front; organize the set of possible values, but hide them nearby if they take up too much space.

Table 10- Problem/Solution Summary of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (part 7)

No	Name	Problem	Solution
38	Sliding Scale	How should the artifact indicate what kind of information should be supplied?	Show the range of values visually, as with a line or arc; show the current value as a location in that line or arc, and if the value is set-able by the user, make that location directly changeable by the user, as with a slider or knob – or by simply touching or clicking on the desired value.
39	Editable Collection	How should the artifact indicate what the user is supposed to do with that collection?	Show the collection to the user, along with obvious ways to remove or change the position of each item. To add an item, make it eminently clear whether the user should obtain the item before or after the "add" command or gesture.
40	Forgiving Text Entry	How does the artifact indicate what kind of information should be supplied?	Allow the user to enter text in any recognizable format for that context.
41	Structured Text Entry	How does the artifact indicate what kind of information should be supplied?	Rather than letting a user enter information into a blank and featureless text field, put structure into that text field
42	Toolbox	How should the artifact present the actions that the user may take?	Keep tools together, put them physically close to the user's working surface, and make sure they are distinct from other actions that the user may take.
8- How can the user modify the artifact?			
43	User Preferences	How does the artifact present the actions that the user may take?	Provide a place or working surface where users can pick their own settings for things like language, fonts, icons, color schemes, and use of sound.
44	Personal Object Space	How should the items in question be organized?	Allow users to place things where they want, at least in one dimension but preferably in two.

Table 10- Problem/Solution Summary of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (part 8)

No	Name	Problem	Solution
45	Scripted Action Sequence	How can the artifact make repetitive tasks easier for the user?	Provide a way for the user to "record" a sequence of actions of their choice, and a way to easily "play them back" at any time.
46	User's Annotations	How can the artifact help preserve the user's hard-won understanding from one use session to the next?	Support ways for users to add their own comments and other annotations to the artifact.
47	Bookmarks	How can the artifact support the user's need to navigate through it in ways not directly supported by the artifact's structure?	Let the user make a record of their points of interest, so that they can easily go back to them later.
48 Iconic Reference.			
49 Calm Grid.			
50	Repeated Framework	How should the content be presented in a unified and consistent way, so that the user can easily navigate through it and quickly become familiar with it?	Design a simple, flexible visual framework for the content, then repeat it on every page or working surface; position the content within that framework, allowing the form of the content to vary as needed.
51 Few Hues, Many Values.			
10- How else can the artifact actively support the user?			
52	Good Defaults	How does the artifact indicate what kind of information should be supplied?	Supply reasonable default values for the fields in question.

Table 10- Problem/Solution Summery of A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell (part 9)

No	Name	Problem	Solution
54	Interaction History	Should the artifact keep track of what the user does with it? If so, how?	Record the sequence of interactions as a "history."
55	Progress Indicator	How can the artifact show its current state to the user, so that the user can best understand what is going on and act on that knowledge?	Show the user a status display of some kind, indicating how far along the process is in real time.
56	Important Message	How should the artifact convey this information to the user?	Interrupt whatever the user is doing with the message, using both sight and sound if possible.
57	Reality Check	How can the artifact protect itself and the user from these kinds of actions, while allowing the user to have the final say over whether or not an action is performed?	Before the action is performed, tell the user what the side effects of the action will be, and ask the user to confirm that's what they really want to do.
58	Demonstration	How can the user learn how to use the artifact?	<i>Demonstrate how to do it.</i>
59	Quick Access		
60	Familiar Quantity		

I used some of her patterns in my proto pattern language as the patterns for the widgets of an informative artifact. Table 10 shows a Problem-Solution Summery of Tidwell's pattern language. I used only those parts of the solutions in her patterns that she highlighted herself.

Iconic Reference Pattern

I wrote a pattern of these pattern language as an assignment for my pattern course in Fall 1998. ¹⁸¹ I tried to write this pattern as a piece of a puzzle to fit exactly in the frame work of Tidwell Pattern Language. [Azad, 98] The Problem / Solution summary of this pattern is shown in Table 11.

Table 11- Problem/Solution Summary of Iconic Reference Pattern

No	Name	Problem	Solution
48	Iconic Reference Pattern	How can you reduce the language dependency, and minimize the undesired repetition of description in an artifact?	Create an Icon with these attributes: Simple, Readable, Stylized, Related to the Message, Distinguishable, Composeable, Culturally Transferable, Interactive, Standard. Test your Icon in different ways with different users.

*A Proto Pattern Language for Human-Computer Interaction***Appendix 5: Section 7:****Joint Application Development and
Participatory Design**

What is Joint Application Development?

Phases of Joint Application Development

Advantages of Joint Application Development

What is Participatory Design?

Motivations of Participatory Design

Goals of Participatory Design

Advantages of Participatory Design

Comparing Participatory Design and Joint Application Development

Combination of the two Methodologies

Conclusion

*A Proto Pattern Language for Human-Computer Interaction***Appendix 5: Section 7:****Joint Application Development and Participatory Design**

Joint Application Development (JAD) and Participatory Design (PD) are two well-known methods for developing systems with more user involvement and user participation. They have been well established in North America and Scandinavia respectively. Both of them focus on the interaction between users and designers wherein dynamic group techniques are employed for eliciting and refining ideas. ¹⁵²

What is Joint Application development?

Joint Application Development was originated in IBM in the late 1970's. Traditionally, Joint Application Development has been a joint venture between users and data processing professionals. In recent years, it has become a joint venture among the people who need to make decisions affecting multiple areas of an organization. It is a structured workshop (usually called Joint Application Development Session) where people come together to plan projects, design systems, or make business decisions. Each Joint Application Development session has a detailed agenda, visual aids, a facilitator who moderates the session, and a scribe who records the agreed-upon requirements. A final

document containing all the decisions made by the group is generated after each session.

A Joint Application Development session is a facilitated session. Joint Application Development was originally designed to address information system development. A Joint Application Development session usually involves some aspects of system design, or development. The use of Joint Application Development techniques has expanded to handle a broader range of challenges. Nevertheless, all the sessions are still facilitated sessions which are led by facilitators.

Joint Application Development sessions have many other names as follows: Accelerated Design, Facilitated Meetings, Facilitated Sessions, Facilitated Team Techniques, Facilitated Work Sessions, Group Design, Interactive Design, Interactive Joint Application Development, Joint Application Design, Joint Requirements Planning, Joint Sessions, User Centered Design.

Phases of Joint Application Development

Joint Application Development has five phases. It evolves from a definite start to a clear finish. The five phases are as follows:

Phase 1: *Project Definition*. Through this phase you define the project purpose, scope, and objectives, identify Joint Application Development team members, and set schedules.

Phase 2: *Research*. Through this phase you gather more details about

the user requirements, you explore the problem domain, and consider design issues. Based on the research, you prepare an agenda listing what needs to be decided in the session.

Phase 3: *Preparation*. Through this phase you prepare everything you need for the session, such as visual aids, working documents, flip charts, overhead transparencies.

Phase 4: *The Session*. This is the actual workshop session. Through this phase Joint Application Development team members define business process, and project requirements. Agreed-upon decisions are documented for the final document.

Phase 5: *The Final Document*. Through this phase the information captured in the session is used to produce the final document. This document is the final product of a Joint Application Development session.

Following figures (45 - 49) illustrate the five phases.

Figure 45:

Figure 46: **Figure 47:** **Figure 48:**

Figure 49:

Advantages of Joint Application Development

Since the first session in IBM, Joint Application Development has been adopted by all sizes of companies around the world. Companies nowadays are looking for systems with flexible functions, less budgets, a tighter schedule. Faced with these constraints, traditional methodologies, such as a waterfall life cycle, don't work well. Joint Application Development sessions let designers and users work together in the very early stage of the development cycle, defining the scope, requirements of projects, and resolving conflicts among different user groups. It puts much effort early in the life cycle in order to improve the quality and increase productivity. The following trends further encouraged the fast adoption of Joint Application Development :

Group Work: Group work can achieve enhanced focus, task orientation, innovativeness and individual commitment. A group can have more

dynamics than individuals working along in a traditional hierarchy structure.

The emphasis in corporate structure is shifting away from hierarchy and more toward group work. Joint Application Development , by itself, is organized as group work.

Focus on Quality and Productivity: Companies are seeking for quality products and high productivity. Quality can have many means. The most frequently cited success indicators for applications were: *timely delivery, achievement of project objectives, and building positive communications and relationships with customers*. In the meantime, people are expecting higher productivity. Joint Application Development put much emphasis on users ensuring that the product is what the customer wanted.

Shift from Technology to Business: Technology is just a means which should serve the business needs. No matter how advanced a technology is, it would be useless if it cannot serve the business requirements and improve business process. Joint Application Development involves users in the development life cycle and lets users define their requirements.

Thus, it ensures that the system developed satisfies the actual activities of the business.

What is Participatory Design?

Traditionally democratic values which have been taken into account in the

design process have only been those concerning technical and economic factors. But the socio-technical approach which began in England by Enid Mumford,¹⁵³ and the collective resource approach developed in Scandinavia by Pelle Ehn¹⁵⁴ and Morten Kyng¹⁵⁵ challenge this design tradition [Bjerknes, et al, 87]. The result of applying the socio-technical approach in Scandinavia is the developing of Participatory Design (PD) in late 1970's. It is pro workplace democracy in the context of technological growth and business development (direct and effective worker participation in design activities and decisions, within a trade union context.) Now the area of Participatory Design has been growing rapidly, in terms of numbers of practices, extent of theoretical development, numbers of practitioners, and geographical and institutional diversity of practice. Outside of Scandinavia, the theorists and practitioners use it to pursue a locally adapted form of democratic decision-making, and others emphasize effective knowledge acquisition and product quality.

Design of computer support is design of labor processes. The user is experienced in the application domain and knows how to play the "language games" in the work or use situations. Similarly designers are experienced in the "language games" of systems design. Both aspects are important. Design should be done with users, neither for nor by them.¹⁵⁶ Mutual learning should be an important part of the work in a design group which is not merely a visiting

workplace or research laboratory. It should be more design-oriented activities.

Design by doing is such a way that considers the labor process not only as an information process but also reflects the democratic aspects of the labor processes and computer support. Users can articulate their demands and wishes in a concrete way by actually doing simulated work. Computer technology has brought changes to the work situation. So techniques used must be designed with thought for their human consequences and introduced with the attempt to involve or consult with the employees who will be using them. According to these considerations, Participatory Design embodies two main principles as follows:

[Miller, 93]

1- workers and customers are intelligent, creative, and productive contributors to organizations if they are empowered to express their insights, apply their expertise, exercise their decision making capabilities, and given responsibility for the impact of their actions.

2- Participatory Design holds that good ideas are as likely to come from the bottom up as from the top down.

Motivations of Participatory Design

According to the above description of Participatory Design, the motivations for using Participatory Design can be characterized as follows:

1- Workers should be given better tools instead of having their work or

their skills automated.

2- Users are qualified to determine how to improve their work and their work life.

3- Users' perceptions and feelings about technology are as important as technical specifications or performance indices.

4- Information technology can only be appropriately addressed within the context of the workplace.

Goals of Participatory Design.

The primary goals of Participatory Design are the involvement of the individuals who do work in a process in the designing of that process and the building of shared understanding and knowledge amongst the participants in a process. Participatory Design recognizes that design is fundamentally a social activity which is not entirely rational and that it is a process of reflection-in-acting best done in an iterative or hermeneutic manner.

Advantages of Participatory Design.

The advantages of using Participatory Design are as follows: [Harris, et al., 96]

1- Involving users of different organizational groups and providing a forum in which the participants can better understand boundaries and explore ways to overcome them.

2- Shared understanding and knowledge of the process enable individuals

to anticipate the consequences of their actions.

3- Increasing workers' influence on technological change

4- Participation in the design process also prepares people for changes

5- Participation was used as a tool to open up conflicts and negotiated throughout the design processes

Techniques of Participatory Design.

Some of the techniques can be used in Participatory Design are as follows:

[Carmel, et al., 93]

Visualizing the current workplace: immersion of designers and facilitators in the target workplace (hands-on apprenticeship), games (structured activities and interactions)

Visualizing the possible workplace: future workshops, metaphor-based design site visits, storyboard, video productions, brainstorming, improvisational theater and role-playing, and various types of graphic illustration.

Prototyping: Presentation and Evaluation of concrete options:

Cooperative prototyping, Props and mockups of available materials (cardboard). The users actually work with a prototype and experience it.

From above we can see that Participatory Design is a complex process involving technology and multiple levels of organization. It is also highly

dependent on specific organizational contexts. For project participants, this means there is no programmatic solution. To the extent that Participatory Design projects have been documented to date, there is little evidence that a "standard" set or ordering of practices has been decided. However, Participatory Design efforts in recent years have led to a repertoire of flexible practices and general guidelines.

Figure 50 shows a brief guide to Participatory Design practices.

Figure 50: A brief guide to Participatory Design practices

To read figure 50 you should notice the following description of the notions that are used in this figure:

Time during the development life cycle: some practices appear to be more appropriate at certain points within the development life cycle or iteration. The horizontal axis of the figure provides a very approximate guide to points within the life cycle at which each practice may be useful.

Who participates with whom in what? The concept of participation is open to multiple interpretations. The vertical axis of the figure spans one way of organizing the various approaches, the software professionals can participate in the users' world lower on the axis, or users can participate in the software professionals' world (higher on the axis).

Appropriate group size for the practice: Different practices are designed to work with groups of different sizes. Appropriate group sizes are indicated by superscript letters for each category of practice: T (tiny, 2-4 participants), S (small, 6-8 participants), M (moderate, up to 40 participants), and B (big, up to 200 participants). The group size recommendations are in some cases approximate.

Comparing Participatory Design and Joint Application Development

As I mentioned, the methods of Joint Application Design and Participatory Design have established themselves in North America and Scandinavia

respectively. They are well-known approaches for operationalizing user involvement and user participation. [Carmel, et.al, 93] Both of them focus on facilitated interactions between users and designers wherein dynamic group techniques are employed for eliciting and refining ideas. However, due to the differences on their socio-political contexts, there are several contrasts between the two methodologies. As Carmel and others [Carmel, et al, 93] stressed, Participatory Design and Joint Application Development are simultaneously similar, complementary, and contradictory. Consequently, a careful analysis and comparison would benefit those who teach and work in information systems development.

Similarities:

User involvement: The main common characteristic of Participatory Design and Joint Application Development is that they advocate a strong user involvement in system design, in which users and workers actively engage in designing the computer system they will eventually use. The Information Systems (IS) community subscribes to the notion that the success of a system is proportional to the degree to which the users of that system are involved in its design and development. [Carmel, et al., 93]

Modalities of user involvement: Both methods involve the users in workshops that encourage creativity and new thinking. Practitioners in

both methodologies employ simple, low-level documentation and visualization methods in their workshops in order to improve the communication and collaboration with the user.

Obstacles: Both approaches face the same obstacles in successful implementation: managerial resistance, user conservatism, lack of user workshops and poor facilitators. Getting user participation is always a test of perseverance: the managers are too busy the workers are not given approval to spend much time away from their jobs. On the other hand the users themselves can be uncooperative and unmotivated.

Contrasts:

Even if the two methods address the same issue of user involvement in the design of the system, there are several differences between them.

These contrasts come from the fact that Participatory Design emphasizes the social aspect of collaborating with the user, meanwhile, Joint Application Development, or "the North American approach of Participatory Design," is based on functional focuses with fixed requirements, communication through documentation and rules of work.

The differences between the two methods are summarized in Table 12.

Subject	Joint Application Development	Participatory Design
Criteria for validation	Quantitative: economic optima, performance indices, time savings	Qualitative: democracy, mutual learning, mutual education, conflict resolution
Background / theory	Group dynamics, software engineering	Labor relations, group learning
Goal	Improved system	Improved workplace
Roots	Industry Canada, USA	Governments, unions, academe Scandinavia
Current practice	Consultancy for profit	Consultancy on principle
Themes	Teamwork, accelerated design, completeness	Democracy of the workplace, social context, industrial democracy, empowerment, humanization
Focal activity	The meeting : - delimited by time - set agenda	Group process: - satisfaction delimited - agenda negotiable
Techniques' emphasis	Structure	Creativity
Perspectives on users	Both operational workers and managers are considered "users" User selection based on competence criteria Users are viewed as one source of knowledge	"Users" are the operational /users. Managers are grouped separately User participation is mandatory Users are viewed as primary source of knowledge

Table 14: A summary of contrasts between Joint Application Development and Participatory Design [Carmel, et al., 93]

Combination of two Methodologies.

Some people who are aware of the weaknesses of Joint Application Design methodology suggest the use of a combination of both. [Carmel, et al., 93]

According to this proposal regardless what method was chosen, it can benefit from ideas implemented by the other one. Being aware of the weaknesses of the method you are using can benefit you by introducing principles of another method which can solve some problems. In other words someone can "tailor" a Participatory Design or a Joint Application Development method to be more suitable to her needs by combining some principles. Example areas in which the two methods can benefit from each other are presented in Participatory Design and Joint Application Design: A transatlantic Comparison. [Carmel, et al., 93] Joint Application Development, for example, can benefit from Participatory Design in following areas:

The user participant. One of the problems in Joint Application Development is that the operational employees are overlooked as participants in Joint Application Development meetings because of an unjustified lack of confidence that front-line workers can meaningfully contribute to the design process. Consequently, this results in a meeting room filled with middle managers and supervisors unable to specify details of day to day operations. If Participatory Design ideal benefits would be considered, this would provide a basis for opening up Joint Application Development to increased worker participation.

Creativity. Although Joint Application Development practitioners utilize

many creative techniques and paraphernalia in the design process, it was found that often these are minimal. Therefore, it is perhaps difficult for many practitioners to be creative in a Joint Application Development workshop. On the other hand Participatory Design practitioners tend to display a flair for creativity that many people in the systems development field simply do not exhibit. This suggests that Joint Application Development 's creative potential can be enhanced through facilitator creativity training.

There are also areas where Participatory Design practitioners can benefit from Joint Application Development principles:

Structure. Joint Application Development method emphasizes structure, while the Participatory Design approach devotes almost no guidelines to structure. A rigid structure method does not facilitate creativity (as Participatory Design practitioners claim). However, it was proved that a structure which preserves the contextual flexibility and Participatory Design practitioners consider it important, can actually enhance creativity.

Moreover, a guideline for using Participatory Design method will push this method further into the hands of the average designer system analyst in industry.

These are not the only areas that the two methods can benefit from the other.

A close analysis of their similarities and contrasts may reveal other aspects in which the two methods are complementary. Similarities suggest a basis for future mutual development, while contrasts suggest points of mutual learning.

Conclusion

Joint Application Development is a structured approach. It has well-defined five phases which every Joint Application Development session should include. The Joint Application Development session is facilitated by the facilitator who controls the pace of the session. There is an agenda for each session which sets the goals and issues that the session should accomplish. The focus of the Joint Application Development is more on economic and performance.

Participatory Design recognizes that design is fundamentally a social activity, is not entirely rational, and that it is a process of reflection-in-acting best done in an iterative manner. The focus of Participatory Design is to improve workplaces. It emphasizes more on the social effect of the system. To claim that there is no "standard" or fixed order of practices in Participatory Design that one can follow, is an aspect of a having a static and reactive attitude(in contrast to dynamic and proactive approach of Participatory Design) toward the system design.

Alexander's pattern theory, and pattern movement (in architecture or software), follow a participatory design principle. They look at the system as a social-technical system. Joint Application Design defines a system as a technical-

social system. Therefore, in spite of any claim to define itself as "User Centric Design", in reality for Joint Application Design approach, performance of the system has priority to the user needs. This approach is more expensive and time consuming and less effective than Participatory Design. For these reasons application of Joint Application Development seems to lose its ground and popularity soon.

A Proto Pattern Language for Human-Computer Interaction

Notes, References, Glossary

Notes
References
Glossary

A Proto Pattern Language for Human-Computer Interaction

Notes

Chapter 1: Morphology of the Concept

1- For more information about Critical Social Theory see: **Andrew Freenberg's** publications, including [Freenberg, 91].

2- Karl Marx 1818-83,

German social philosopher and revolutionary, a founder of modern Socialism and Communism. He rejected the idealism of Hegel but was influenced by Ludwig Feuerbach and Moses Hess.

With Engels he wrote the Communist Manifesto (1848) and other works that broke with the tradition of appealing to natural rights to justify social reform, invoking instead the laws of history leading inevitably to the triumph of the working class.

Exiled from Europe after the Revolutions of 1848, Marx lived in London, while working on his monumental work *Das Kapital* (3 vol., 1867-94), in which he used Dialectical Materialism to analyze economic and social history. Engels edited vol. 2 and 3 after Marx's death. With Engels, Marx helped found (1864) the International Workingmen's Association, but his disputes with the anarchist Mikhail Bakunin eventually led to its breakup.

Marxism has greatly influenced the development of socialist thought. Further, many scholars have considered Marx a great economic theoretician and the founder of economic history and sociology. [Encyclopedia, 2000]

3- The Institute of Interdisciplinary Studies

Carleton University, Ottawa, ON, Canada

<http://www.carleton.ca/iis/>

4- Mosco, Vincent

The School of Journalism and mass communication

Carleton University, Ottawa, ON, Canada

<http://www.carleton.ca/jmc/mass/faculty/mosco.htm>

5- The School of Journalism and mass communication
 Carleton University, Ottawa, ON, Canada
<http://www.carleton.ca/jmc/>

6- This is a reference to the concepts of Desk-killing, Stratification, and Job Loss.

7- This conclusion is extracted mainly from: [Booker, et al., 97]

8- I owe my awareness of the application of the pattern theory in the software design to Dr. Dwight Deugo and his course (Special Topics in Computer Science, School of Computer Science, Carleton university, Fall 1998)

9- Pablo Picasso, (Pablo Ruiz Picasso), 1881-1973
 Spanish painter, sculptor, graphic artist, and ceramicist who worked in France. He was the foremost figure in 20th-cent. art. Leader of the School of Paris, he was remarkable for his technical virtuosity, incredible originality, and prolificacy. Admitted to the Royal Academy of Barcelona at 15, he later moved to Paris, where he remained until 1947, then moving to the South of France. His early works, e.g., Old Woman (1901; Philadelphia Mus. Art) show the influence of Toulouse Lautrec. His production is usually described in series of overlapping periods. In his melancholy blue period such works as The Old Guitarist (1903; Art Inst., Chicago) depicted, in blue tones, the world of the poor. His rose period is characterized by a lighter palette and subjects from the circus. In 1907, Picasso painted Les Femmes d'Alger (O.J. version O) (Mus. Mod. Art, N.Y.C.), the most significant work in the development of Cubism and abstraction, and a herald of analytic cubism. In the synthetic phase of cubism (after 1912), his forms became larger and more representational, e.g., The Three Musicians (1921; Mus. Mod. Art, N.Y.C.). In the 1920s he also introduced Collage. His second landmark work was Guernica (Reina Sofia, Madrid; Centro de Arte Reina Sofia), an impassioned condemnation of war and fascism. In his later years, Picasso turned to creations of fantasy and comic invention. Working consistently in sculpture, ceramics, and the graphic arts, he continued to explore his personal vision until his death at 91. [Encyclopedia, 2000]

10- Institutional Architecture or Public Architecture as classified by professional bodies of architecture in North America.

Chapter 2 Current Challenges in the field of Human-Computer Interaction

11- Chapter 2 (Current Challenges in the field of Human-Computer Interaction) is extracted mainly from: [Myers, et al., 96]

For previous reports on Human-Computer Interaction, see: [Foley, et al., 95], [Sibert, et al., 93], and [Strong, 93]

12- Some examples of decreasing the cost by applying Human-Computer Interaction's techniques can be found in [Karat, 90], [Mantei, et al., 88], [Gray, et al., 93], and [Nielsen, et al., 93]

Karat reported savings due to the use of usability engineering of \$41,700 in a small application used by 23,000 marketing personnel, and \$6,800,000 for a large business application used by 240,000 employees. Savings were attributed to decreased task time, fewer errors, greatly reduced user disruption, reduced burdens on support staff, elimination of training, and avoidance of changes in software after release.

Mantei and Teorey reported the mean benefit for finding each usability problem at \$19,300.

Gray, John, and Atwood reported that a proposed workstation saved a telephone company \$2 million per year in operating costs.

Nielsen and Landauer reported that a mathematical model that using software that had undergone thorough usability engineering will save a small project \$39,000, a medium project \$613,000 and a large project \$8,200,000. Therefore, by estimating all the costs associated with usability engineering, the benefits can be up to 5000 times the cost.

13- Some examples of the catastrophes resulted from using an insufficient interface can be found in [Neumann, 91], [Ladkin, 96], and [Leveson, et al., 93]

Neumann reported that the complicated user interface of the Aegis tracking system was a contributing cause to the erroneous downing of an Iranian passenger plane. Also, the US Stark's inability to cope with Iraqi Exocet missiles was partly attributed to the human-computer interface.

Ladkin reported that problems with the interfaces of military and commercial airplane cockpits have been named as a likely cause for several crashes, including the Cali crash of December 1995.

Leveson and Turner reported that a number of people died of radiation overdoses partially as a result of faulty cursor handling code in the Therac-25.

14- For a complete history of the technological developments in the field of Human-Computer Interaction see [Myers, 96]

15- Figure 2 is borrowed from [Myers, et al., 96]

16- Figure 3 is borrowed from [ACM, 92]

17- For more information about "Information Visualization" see: [Catarci, et al., 96]

18- For more information about social-technical systems see [DISE, 98]. DISE (Department of Industrial and Systems Engineering - Virginia Polytechnic Institute and State University) under the title of **Macroergonomics and Sociotechnical Systems Theory** defines a socio -technical system as follows:

"The term Sociotechnical System, or STS, was first introduced in the 1960's by researchers Emery and Trist to describe the complex nature of large human-machine organizational systems. Sociotechnical systems theory is characterized by several core principles: joint causation, joint optimization, and joint design. Joint causation refers to the notion that the social and technical subsystems are both affected by the environment. Since both subsystems respond to causes in the environment, optimizing one suboptimizes the other. Joint optimization is therefore the goal. Joint design is the method by which joint optimization is achieved."

For recent social-technical viewpoints of system design see [Bowker, et al., 97].

Chapter 3: Standards and/or Patterns

19- Chapter 3 (Standards and/or Patterns) is extracted mainly from: [Ameritech, 98], [Appleton, 97], and [Deugo, 98]

20- Chomsky, Noam, 1928 -

During 1945-50, Chomsky was a student of the University of Pennsylvania where he began his study of linguistics. He is currently a prominent American linguist at Massachusetts Institute of Technology (M.I.T) and a political activist.

He considers only one narrow aspect of human language, syntax. The early days of computing programs in the late '50s were focused on translating one natural language into another and were seen as a potential solution to the "worldwide translation problem." This is when scientists came to understand that there is more to language than meets the eye. This is where Chomsky entered the picture; his knowledge of linguistics and syntax contributed the theoretical basis for programming to computer science.

Chomsky, a libertarian socialist, is acknowledged to be an articulate and persuasive critic of U.S. foreign policy. A fundamental aspect of his critique is the crucial role that U.S. academics and journalists play in the "manufacture of consent." His writings in this area include books on the Vietnam War, Middle East, and on terrorism.

21- Peter Eisenman, 1932-

Peter Eisenman was born in Newark, New Jersey in 1932. He studied at Cornell and Columbia Universities and then at Cambridge University in England. He taught at Cambridge, Princeton and the Cooper Union in New York, where he was founder and director of the Institute for Architecture and Urban Studies.

Until recently, few of his designs had been built. As a result, most attention has focused on his architectural ideas which attempt to create contextually disconnected architecture.

Eisenman has always sought somewhat obscure parallels between his architectural works and philosophical or literary theory. His earlier houses were "generated" from a transformation of forms related to the tenuous relationship of language to an underlying structure.

Eisenman's latter works show a sympathy with the "anti-humanist" ideas of deconstructionism. [Sharp, 91]

For more info on Eisenstein see: [Willoughby, 93], [Achten, 94], [Blank, 98] [Burks , 95]

23- For a description of the role of scenarios, use cases, and the process of capturing requirements in the process of Human computer interaction design in Joint Application process see [IBM, 2000], [Microsoft, 2000], and [Berkun, 99].

Chapter 4: A proto-Pattern Language for Human-Computer Interaction

23- For more information about the pattern based architecture see: Appendix 1 (Patterns in Architecture)

24- Metaphor is different from modeling. For the definition of Metaphor in the scope of my thesis and Modeling see: Glossary.

For more information about metaphorical view to the space of an artifact see: [Rohrer, 95]

25- For more information about "Alexander's Pattern Theory" see: Appendix 2 (Alexander's Pattern Theory)

26- For more information about "WindowPerTask pattern" see: Part 3 of Appendix 5 (Window and its Patterns)

27- For more information about software's patterns see: Appendix 3 (Patterns in Software)

28- For more information about patterns in the field of human-computer interaction see: Appendix 4 (Patterns in Human-Computer Interaction)

29- Part 3 of Section 1 of Chapter 4 (Format) is extracted mainly from:

[Appleton, 97], and [Deugo, 98]

30- Part 4 of Section 1 of Chapter 4 (Components of a Pattern) is extracted mainly from: [Appleton, 97], and [Deugo, 98]

31- For more information about the component of Forces see: Part 5 of Section 1 of Chapter 4 (Forces)

32- For more information about the component of Sketch see: Part 6 of Section 1 of Chapter 4 (Background: Sketch)

33- Part 5 of Section 1 of Chapter 4 (Forces) is extracted mainly from: [Appleton, 97], and [Deugo, 98]

34- By informative artifact, I mean "An artifact that is giving information," for example: " an informative book."

For more information about "Informative" as a term, see: the Glossary and [infoplease-3, 99]

35- Part 6 of Section 1 of Chapter 4 (Sketch) is extracted mainly from: [Coplein, 99-1]

36- The bolded terms are equivalent of the concepts in the vocabulary of the pattern based architecture.

37- My proto-pattern language follows the same format that is described in: "A pattern Language for Pattern Writing." [Mezaros, et al., 97]

38- See Appendix1 (Patterns in Architecture.)

39- See Appendix1 (Patterns in Architecture)

40- See Appendix1 (Patterns in Architecture)

41- See Appendix1 (Patterns in Architecture)

42- This is a representation of one of the main principles of the system theory at the interaction domain. This principle conveys that "The manager of the system is the most complicated member of the system."

43- The concept of the "Accessibility" in architectural and urban spaces is described in the Rationale heading of this pattern.

44- Masha and Seethaler [Mascha & Seethaler, 96] in their proposal "Relaxation of controlling mechanisms by deregulation," which was introduced to "Millennium workshop 96" in Vienna, reported that:

" According to the selected parameters a multiplicity of possible and applicable solutions (zoning-formations) exist for one site . . . "

45- Deregulation of the content and its effect on reducing the complexity is explained in Appendix 5 Section 2 (Organization and Complexity.)

46- A miniature does reflect only those inherited information from its model that can be represented in its scale. A miniature is not a copy of the model. The same concept can be seen in the relation between thread and process. Thread is a miniature of the process.

47- This solution is an application of peace meal growth. For more information about the concept of peace meal growth see: [Alexander, et al., 75], [Appleton, 97].

48- Watanabe in Kolmogorov Complexity and Computational Complexity [Watanabe, et al., 92] explains that:

Complexity of process, e.g., "Computational complexity, usually a complexity measure for a set of objects, measures the computational resources necessary to recognize or produce elements of the set."

49- Uspensky in Kolmogorov Complexity and Computational Complexity [Watanabe, et al., 92] explains that:

"Complexity of things (as opposed to the complexity of processes, e.g., of computational processes) took the name descriptonal complexity, or Kolmogorov complexity . . . in appropriate cases one may say entropy instead of complexity."

50- Uspensky in Kolmogorov Complexity and Computational Complexity [Watanabe, et al., 92] explains that:

" Things can be large or small, and their size (the length or the volume or the weight or so on) can be measured by a number, Besides, things can be simple or complex, and their complexity can also be measured by a number."

51- Masha and Seethaler [Mascha & Seethaler, 96] in their proposal "Relaxation of controlling mechanisms by deregulation. " that was introduced to "Millennium workshop 96" in Vienna, explained that:

" Dynamic zoning-matrix (instead of the existing rigid and over regulated zoning-plan) is going to cope with relevant developments and demands of the current and future society. How can a society be prepared for different and unpredictable developments in urban planning? A complexity, such as special demands and different lifestyles of individuals or groups, can only be articulated in an open system, the matrix provides. The logistic expression of such a matrix is the so-called zoning-formation, a deduction from the matrix. Influence on the matrix can only be taken by concrete and practical rules; that is the importance and the conation of relevant topics in their significance for a specific site or area. That guarantees, that the intentions, or the will of a certain group or individual, flows into the rules and secures the special demands, not in the actual form, which is often bound to archaic thought. The formal result is deprived of all "will" concerning the architectural form, leaving more room to move. Controlling mechanisms that always have to strive to archive synchronicity and uniformity are overruled by a systematic approach that only serves practical and concrete demands. Deregulation together with the guarantee of an open process development and the security of a demand-fulfilling result are thus implemented into an archaic system of formal aesthetics, which does not serve any modern and diverse society."

52- As shown in example heading of *Space Pattern* of this Proto-Pattern Language

53- For the definition of transparency principle see: Chapter 4, Section 1, Part 5: (Background: Forces)

54- As shown in the example heading of the *Space Pattern* of the proto-pattern language.

55- As shown in the example heading of the *Grading Pattern* of the proto-pattern language.

56- See the sketch heading of this pattern for a graphic representation of an ordered sequence.

57- You can find a brief of Foote's proposal "A Fractal Model of the Life-cycle of Reusable Objects" [Foote, 91] in Chapter 4, Section 1, Part 2 (Background: Approach).

58- As shown in the example heading of the *Space Pattern* of the proto-pattern language.

59- As shown in the example heading of the *Grading Pattern* of the proto-pattern language.

60- As shown in the example heading of the *Navigation Pattern* of the proto-pattern language.

61- As shown in the example heading of the *Space Pattern* of the proto-pattern language.

62- As shown in the example heading of the *Grading Pattern* of the proto-pattern language.

63- As shown in the example heading of the *Navigation Pattern* of the proto-pattern language.

64- in Cinematography a transition is a link between two sequences that supports their conceptual relation to each other. Cinematographic transitions do not support consistency or organization of a product as a whole.

65- As shown in the example heading of the *Space Pattern* of the proto-pattern language.

66- As shown in the example heading of the *Grading Pattern* of the proto-pattern language.

67- As shown in the example heading of the *Navigation Pattern* of the proto-pattern language.

68- As shown in the example heading of the *Navigation Pattern* of the proto-pattern language.

69- As shown in the example heading of the *Navigation Pattern* of the proto-pattern language.

70- The term "Protected" is a reference to the concept of protected methods in the object-oriented computer languages.

71- These transitions include $n*(n-1) = 20$ hierarchical relationships plus 3 horizontal relationships, as described in figure 51:

Figure 51: The hierarchical and horizontal relationships in an informative artifact (Educational CD)

72- See Chapter 5, Section 1 (Patterns in Architecture) for a description of

formalism and formalistic view.

73- See the example heading of this pattern.

74- Lemon and others in "Cognitive Architectures, A hypertext analysis of Architectures for Intelligence" [Lemon, et al., 99] explained that:

"Frames may indicate specific values for some attributes or instead indicate a default. This is especially useful when values are not always known but can generally be assumed to be true for most of the class. For example, the class BIRD may have a default value of FLIES set to TRUE even though instances below it (say, for example, an OSTRICH) have FLIES values of FALSE."

According to the previous document, the following architectures in the domain of artificial intelligence use a frame-like representation:

1. Plan-then-Compile Architecture (THEO)
2. Meta-Reasoning Architecture (MAX)
3. A Basic Integrated Agent (Homer)
4. Problem Solving Architecture (Soar)

75- See the technical background of this pattern in: Appendix 5, Section 5, Part 3 (State and its Patterns).

76- As shown in the example heading of the *Space Pattern* of the proto-pattern language.

77- As shown in the example heading of the *Grading Pattern* of the proto-pattern language.

78- As shown in the example heading of the *Navigation Pattern* of the proto-pattern language.

79- As shown in the example heading of the *Sequential Pattern* of the proto-pattern language.

80- As shown in the example heading of the *Short-Cut Pattern* of the proto-pattern language.

81- According to statistic techniques the number of samples per group is determined by the size of that group. In the case of an informative artifact the size of the target group of users is not determined, but you may suppose that it is a large group of users. The rate of sampling per a large group is 1 percent. Can you test your artifact with too many users? Your audiences are unknown. Do you know how to find those samples among a large group of unknown people? Using a Beta Version is a good solution, but from the economic point of view this approach cannot be used for small scale productions.

82- See the *Short-Cut Pattern* for more information about "multiple clicks."

83- As shown in the example heading of the *Space Pattern* of the proto-pattern language.

84- As shown in the example heading of the *Grading Pattern* of the proto-pattern language.

85- As shown in the example heading of the *Navigation Pattern* of the proto-pattern language.

86- As shown in the example heading of the *Sequential Pattern* of the proto-pattern language.

87- As shown in the example heading of the *Short-Cut Pattern* of the proto-pattern language.

88- As shown in the example heading of *Space Pattern* of the proto-pattern language

89- For a brief of the technological background of this pattern see: Appendix 5, Section 4 (Related Patterns: Window and its Patterns)

90- For more information about the definition of shape in the pattern based architecture see: Appendix 1 (Patterns in Architecture)

91- For more information about the concept of the window in software design see: Appendix 5, Section 4 (Related Patterns: Window and its Patterns)

92- As shown in the example heading of the *Space Pattern* of the proto-pattern language.

93- For a brief of the technological background of this pattern see: Appendix 5, Section 5 (Related Patterns: Panel and its Patterns)

94- For example you can refer to the works of Piet Mondrian.

Mondrian among the most prominent of the 20th century's geometric painters, evolved an austere art of black lines and colored rectangles placed against white backgrounds. In 1911, Mondrian discovered the cubist works of Pablo Picasso and Georges Braque. Thereafter, in Paris (1912-14), he produced his own version of Cubism. It was to be a very short step from his cubist trees and still lifes to the pure structures that he produced from 1916 onward. Mondrian emigrated to the United States in 1940 and died in New York City four years later. He influenced not only artists slightly younger than himself, such as Ben Nicholson and Max Bill, but also, in the 1950s and '60s, some of the American hard-edge abstractionists. An extensive collection of Mondrian's work is preserved in the Gemeente Museum, The Hague. A rich collection of his works can be find on:

<http://www.usinternet.com/users/glenedwards/links.html>

95- For more information about the definition of the surface in the pattern based architecture see: Appendix 1 (Patterns in Architecture)

96- For more information about the concept of the window in software design see: Appendix 5, Section 4 (Related Patterns: Window and its Patterns)

97- As shown in the example heading of the *Space Pattern* of the proto-pattern language.

98- As shown in the example heading of the *Window Pattern* of the proto-pattern language.

99- For more information about this models and their relationships see: Chapter 1 (Morphology of the Concept and refer to Figure 1: The scope of the proposed proto-pattern language in that chapter.)

100- For a list of these patterns see: Table 11 (List of the published patterns by Plop and EuroPlop that are related to the field of human-computer interaction) in Appendix 4 (Patterns in Human-Computer Interaction.)

101- For a brief review of "A Pattern Language for Human Computer Interaction Design" and "Iconic Reference Pattern" see: Section 6 of Appendix 5 (Related Patterns: Widgets and patterns). See Table 12: Problem/Solution Summary of "A Pattern Language for Human Computer Interaction Design by Jenifer Tidwell - (Part 1 to Part 9), and Table 13: Problem/Solution Summary of Iconic Reference Pattern in that section.

102- As described in the example heading of the *Spatial Pattern* of the proto-pattern language.

103- As described in the example headings of *Organizational Patterns* of the proto-pattern language.

104- As described in the example headings of *Delimiter Patterns* of the proto-pattern language.

105- As described in the example heading of the *Spatial Pattern* of the proto-pattern language.

106- As described in the example headings of *Organizational Patterns* of the proto-pattern language.

107- As described in the example headings of *Delimiter Patterns* of the proto-pattern language.

108- As described in the example heading of the *Conceptual Widgets Pattern* of the proto-pattern language.

109- For more information about Joint Application Development and Participatory Design methodologies see: Appendix 5 Section 7 (Joint Application Development and Participatory Design.)

110- The solution that is documented through this pattern is an application of Participatory Design.

111- The solution that is documented through this pattern is an application of Participatory Design. This pattern is a solution for the question of "How to involve a software designer in the process of documenting a pattern that she is its user."

112- As shown in the example heading of the *Spatial Pattern* of the proto-pattern language.

Chapter 5: Conclusion

113- See the heading of On the Nature of Order in Appendix 5, Section 2, Part 2 (Evaluation of an Idea)

114- See the heading of A Foreshadowing of 21st Century Art; the Color and Geometry of very early Turkish Carpets in Appendix 5, Section 2, Part 2 (Evaluation of an Idea)

115- This is a reference to "SOCIAL SCIENCE, TECHNICAL SYSTEMS AND COOPERATIVE WORK: *Beyond the Great Divide*. [Bowker, et al., 97]

Appendices

Appendix 1: Patterns in Architecture

116- Lea in "Christopher Alexander: An Introduction for Object-Oriented Designers" [Lea, 93] has related this change to "the advent of modern architectural methods." In compare to following statements by Seyyed Hossein Nasr it seems that Lea's conclusions are out of the context [Nasr, 73]. Nasr explains that:

".....those traditional techniques concerning architecture and its components which were once evident to all men but which have become even more forgotten and neglected in the West since the advent of what has come to be known as the Renaissance and among the modernized classes of the East since the spread of the modern mentality from the end of last century. "

117- This Heading (Design paradigm) is extracted mainly from [Lea, 93]

118- In the pattern based architecture form does not necessarily follows the function. Forms, which are temporal, serve as a bridge between the qualitative, abstract world of imagination and the quantitative artifacts of a human person. Symbolic generic forms are capable of many means of physical realization.

Modern architecture paradigm emphasis that **"form follows functions "** and then for modern architecture form has no originality. We should distinguish between the notion of form in the pattern based architectural space, and the notion of form in Formalist Architecture.

The **"form follows function"** is a phrase coined by the Bauhaus movement which describes how the form of an object (it's three-dimensional appearance) is descriptive of how the object is used. For example, if the function of a chair were to support a person in a sitting position, the form would be a horizontal surface and a vertical surface resting on a frame. No embellishments would be added to the form: it would be strictly functional in appearance. For an overview of Bauhaus ideas see: [Fultz, 99]

In pre-modern and post-modern formalist Architecture, form has originality for the sake of form, not based on its potential to define and present its relation to the upper bound abstract system. The modern architectural paradigm of form and function should be considered as an opposition to this notion.

That was an unfortunate destiny that the side effects of this constructive confrontation result in destruction of architectural pattern languages in some other parts of the world. Distinguishing formalist notion from the pattern oriented one is especially important when we bring the idea of patterns to software design, where positivism is a predominant view. Positivism can facilitate the creation of an eclectic point of view.

119- Coplein introduces ritual as follows: [Coplein, 97-2]

"In Preview of the Art of the 21st Century, Alexander describes how truth and beauty are present in ancient Turkish prayer carpets. The artisans who created these carpets were striving to capture the likeness / face of God in them; the structure of the universe."

The concept of ritual is different from Coplien's definition. Ritual does work based on the notion of "active participation in the intellectual and everyday life of the community " and may be formal or actual.

120 -The figures that show parts of the Bazars of Kashan and Esfahan are taken from [Ardalan, et al., 73]

Appendix 2: Christopher Alexander's Pattern Theory

121- Alexander's biography is extracted mainly from: [Salingaros, 97-1] and [DP2, 94]

122- This sentence is extracted from the Jacket of "Pattern Language." [Alexander, et al., 77]

123- Christopher Alexander's list of publications is extracted from the available information on [LC, 99] and [WebCat, 98]

124- The seven publications that are marked by a "*" were not listed in the catalogue of Library of congress.

125-This heading (Domestic Architecture) is an electronic publication. See [Alexander, 94]

126-This heading (Notes on The Synthesis of Form) is extracted mainly from: [Alexander 64] and [Coplien, 97-1]

127-This heading (The Oregon Experiment) is extracted mainly from: [Alexander, et al., 75] and [UOPO-1, 92]

128- The map of Oregon campus is downloaded from [UOPO, 92-2]

129-This heading (A Pattern Language: Towns, Buildings, Construction) is extracted mainly from: [Alexander, et al., 77] and [Saunders, 97]

130-This heading (The Timeless Way of Building) is extracted mainly from: [Alexander 79], [Heather, 98], and [Lea, 93]

131-This heading (A New Theory of Urban Design) is extracted mainly from: [Alexander 86] and [OGC, 99]

132-This heading (A Foreshadowing of 21st Century Art; the Color and Geometry of very early Turkish Carpets) is extracted mainly from: [Alexander 93], [Coplein-2, 97], and [Salingaros, 95]

133-This subheading (The Language of the Carpets) is extracted mainly from: [Tayfun, 97] and [Salingaros-2, 97]

134-This heading (Domestic Architecture) is extracted mainly from: [Alexander 94]

135-This heading (The Nature of Order)is extracted mainly from: [Salingaros, 97-2], [Evans, 96], [Gabriel, 96], and [Coplien, 97-2]

136-This part (Evaluation of Alexander's ideas) is extracted mainly from: [Khare, 95] and [Coplien, 97-2]

137- This sentence is extracted from the Jacket of "Pattern Language." [Alexander, et al., 77]

138- The highlighted words are correspondent terms in Persian Architectural Pattern Language for the concepts that are introduced by Alexander. The most important one is Moment, which are used for the quality without a name. In opposite to the meaning that is implicit in the Alexander's terminology, there is a name for this quality. In Persian language this quality is called "Aan" that is correspondent to Moment in English language. The moment can be defined as "the representation of the knowledge of simultaneous presence of upper bound and lower bound universes at any slot of the time." In this definition Moment is the quality of the Time/Space.

Appendix 3: Patterns in Software

139- This section (Patterns in Software) is extracted mainly from [Appleton, 97] and [Cunningham, 99]

Appendix 4: Patterns in Human-Computer Interaction

140- This section (Patterns in Human-Computer Interaction) is extracted mainly from proceedings of [Plop, 96] to [Plop, 99] and [EuroPlop, 97] to [EuroPlop, 99]

Appendix 5: Related Patterns

141- This part (Organization and Complexity) is extracted mainly from [UNESCO, 97]

142- This part (State and its Patterns) is extracted mainly from: [comp.object, 97], [Houston, 97], and [Perkins, 97]

143- To see the original copy of the State Pattern refer to [Gamma, et al., 95]

144- To see the original and rewritten copies of the WindowPerTask Pattern refer to:

<http://c2.com/cgi-bin/wiki?WindowPerTask>

<http://c2.com/ppr/ui.html#3>

145- For more information about the different MVC's models and their differences see: [Stingray, 99]

146- To see the original copy of the FewPane Pattern refer to:
<http://c2.com/cgi-bin/wiki?FewPanels>

147- Reenskaug Trygve (and others)

Trygve Reenskaug introduced ModelViewController into Smalltalk while visiting XeroxParc. He also wrote one of the articles in the SmalltalkIssueOfByte. Trygve has a book coming out titled Working With Objects: The OOram Software Engineering Method with coauthors P. Wold and O.A.Lehne.

148- This section is extracted mainly from: [Tidwell, 98] and [Azad, 98].

149- I have write down one of these seven unwritten patterns: Iconic Reference Pattern (pattern 48 in Tidwell's set) as an assignment in fall 1998. See [Azad, 98]

150- The course was Special Topics in Computer Science taught by Dr. Dwight Deugo (School of Computer Science, Carleton university, Fall 1998).

151- This section is extracted mainly from [Damian, et al., 99]

152- Enid Mumford is Emeritus Professor at Manchester University, a Fellow of the British Computer Society and a Companion of the Institute of Personnel Management. Previously she was Professor of Organizational Behavior at the Manchester Business School. In 1981 Enid Mumford was awarded the American Warnier Prize for contributions to Information Science. In 1997 she was given an Honorary Doctorate by the University of Jyvuaskyla, Finland.

153- Pelle Ehn is a professor of Malmö University, Sweden. He is responsible for the research at Art and Communication. His current research projects include: What medium will carry the morning news in 2003? Will there be a theater in digitally projected sets? How can the advanced technology be combined with creative ability in innovative projects?

154- Morten kyng is an associate professor at the Computer Science Department of Aarhus University, Denmark. His research interests includes: Cooperative/participatory design; computer support and techniques for cooperative design; and end-user education.

155- This is the same idea that is well described in Domestic Architecture by Alexander [Alexander, 94]. The main misunderstanding about Alexander's works

came from those people who mixed the idea of Participatory Design with the movement of "Architecture without Architects" during the sixties. The core idea of this movement was "Design should be done by users, neither for nor with them." The three different configurations of this sentence resulted in three different understandings of the process of design as follows:

Socio-Technical Systems:

(Participatory Design - and Alexander)

"Design should be done with users, neither for nor by them."

Techno-Social Systems:

(Joint Application Development = User Centric Design)

"Design should be done for users, neither by nor with them."

Architecture without Architect [Rudofsky, 64]

"Design should be done by users, neither for nor with them."

156- One of the best examples of the application of Participatory Design in software design is demonstrated in Martin Fowler's book "Analysis Patterns : Reusable Object Models." [Fowler, 96]

References

Note:

1- The dates that are marked with a star (*), are the dates that I visited related pages. The dates of the creation or the last modification of these pages were not shown.

2- The format of references is developed according to Chicago Style as described in: [UWM, 97]

[Achten, 94]

Achten, Henri.

Review: Cities of Artificial Excavations.

LAVA, 1994

<http://lava.ds.arch.tue.nl/books/bedard.html>

LAVA started in 1993 when a group of students of the Eindhoven University of Technology (TUE) created an Internet site about Architecture. Most of the members from this first period have since graduated.

[ACM, 92]

ACM Special Interest Group on Computer-Human Interaction, Curriculum Development Group.

Curricula for Human-Computer Interaction.

Copyright © 1992 by the Association for Computing Machinery, Inc.

<http://www.acm.org/sigchi/cdg/cdg2.html>

This publication is a report that is prepared by the Curriculum Development Group of the ACM Special Interest Group on Computer- Human Interaction (SIGCHI). Members of the Curriculum Development Group were included Thomas T. Hewett (Drexel University), Chair; Ronald Baecker (University of Toronto); Stuart Card (Xerox PARC); Tom Carey (University of Guelph); Jean Gasen (Virginia Commonwealth University); Marilyn Mantei (University of Toronto); Gary Perlman (The Ohio State University); Gary Strong (Drexel University); William Verplank (ID Two and Stanford University). The Production Editor of this document was Bill Hefley (Carnegie Mellon University), and its electronic version was edited by Gary Perlman (Ohio State University).

[Alexander, 64]

Alexander, Christopher.

Notes On The Synthesis Of Form.

Cambridge: Harvard University Press, 1964

The library of Carleton University has recorded this book as:

Notes On The Synthesis Of Form

Alexander, Christopher. Cambridge, Harvard U.P. 1964 216p.

CALL NUMBER: NK1505.A4 1964

LOCATION (S): FLOOR 3: 2 copies

DESIGN

RN: 282158

<http://www.library.carleton.ca/cube.html>

[Alexander, et al., 75]

Alexander, Christopher; Silverstein, Murray; Angel, Shlomo; Ishikawa, Sara; Abrams, Denny.

The Oregon Experiment.

New York: Oxford University Press, 1975.

The library of Carleton University has recorded this book as:

Oregon experiment.

New York, Oxford U. P. 1975 1 v.

CALL NUMBER: LD4364.6.O73

LOCATION (S): FLOOR 4: 1 copy

UNIVERSITY OF OREGON--BUILDINGS

Alexander, Christopher.

Center for Environmental Structure.

RN: 248117

<http://www.library.carleton.ca/cube.html>

[Alexander, et al., 77]

Alexander, Christopher; Ishikawa, Sara ; Silverstein, Murray; Jacobson, Max; King, Ingrid Fiksdahl; Angel, Shlomo.

A Pattern Language: Towns, Buildings, Construction.

New York: Oxford University Press, 1977.

The library of Carleton University has recorded this book as:

Pattern language: towns, buildings, construction.

Alexander, Christopher. New York, Oxford U. P. 1977 1171 p.

CALL NUMBER: HT166.A6147
 LOCATION (S): FLOOR 4: 2 copies
 CITY PLANNING
 ARCHITECTURE
 ARCHITECTURE--HUMAN FACTORS
 Ishikawa, Sara.
 Silverstein, Murray.
 RN: 202377
<http://www.library.carleton.ca/cube.html>

[Alexander, 79]

Alexander, Christopher.

The Timeless Way of Building.

New York: Oxford University Press, 1979.

The library of Carleton University has recorded this book as:

The Timeless way of building.

Alexander, Christopher. New York, Oxford U.P. 1979 552 p.

CALL NUMBER: NA2500.A45

LOCATION (S): FLOOR 3: 2 copies

ARCHITECTURE

PATTERN PERCEPTION

RN: 505757

<http://www.library.carleton.ca/cube.html>

[Alexander, 84]

Alexander, Christopher.

*The Determination of Components for an Indian Village. In Nigel Cross, ed.,
 Developments in Design Methodology, Chichester.*

John Wiley and Sons, ©1984, 33-56.

[Alexander, et. al., 86]

Alexander, Christopher; Neis, Hansjoachim; Anninou, Artmis; King, Ingrid F.

A New Theory of Urban Design.

Center for Environmental Structure Series, Vol 6,

New York: Oxford University Press, 1986.

The library of Carleton University has recorded this book as:

A New theory of urban design.

Alexander, Christopher. New York, Oxford U. P. 1986 251 p.

CALL NUMBER: NA9031.A38

LOCATION (S): FLOOR 3: 1 copy

CITY PLANNING--PHILOSOPHY

**CITY PLANNING--CALIFORNIA--SAN FRANCISCO--SIMULATION
METHODS**

Neis, Hansjoachim.

Anninou, Artmis.

King, Ingrid F. (Ingrid Fiksdahl), 1937-

RN: 720174

<http://www.library.carleton.ca/cube.html>

[Alexander, 92]

Alexander, Christopher.

A City is Not a Tree.

Republished in *Human Identity in the Urban Environment*. Ed. G. Bell and J. Tyrwhitt, Penguin 1992.

[Alexander, 93]

Alexander, Christopher.

A Foreshadowing of 21st Century Art: The Color and Geometry of Very Early Turkish Carpets.

New York: Oxford University Press, 1993.

[Alexander, 94]

Alexander, Christopher.

Domestic Architecture.

Doors of Perception 2 @HOME Conference,

4-6 November 1994, Amsterdam,

<http://www.mediamatic.nl/Doors/Doors2/Alexander/Alexander-Doors2-E1.html>

[Almanac, 99]

Almanac—Science and Technology—Computers and Internet.

Computer Glossary.

<http://www.infoplease.com/ipa/A0006024.html>

[Ameritech, 98]

Ameritech.

Ameritech User Interface Guidelines.

<http://www.ameritech.com/corporate/testtown/library/standard/index.html>

[Appleton, 97]

Appleton, Brad.

Patterns and Software: Essential Concepts and Terminology.

last modified 11/20/97

<http://www.enteract.com/~bradapp/docs/patterns-intro.html>**[Ardalan, et al., 73]****Ardalan, Nader; Bakhtiar, Laleh.*****The Sense of Unity: The Sufi Tradition in Persian Architecture.***

The university of Chicago Press. 1973

[Autry, 96]**Autry, Roy H.*****What is Organization Design?***<http://www.inovus.com/organiza.htm>**[Azad, 98]****Azad, Minoo.*****Iconic Reference Pattern.***<http://chat.carleton.ca/~mazad2/ReferencePattern.html>**[Beadle, 97]****Beadle, Mike.*****Reengineering the Application Development.***

Process SIGS, New York, summer 1997

[Beck, 95-1]**Beck, Kent .*****Task Window Pattern.***

Copyright 1995, Fiest Class Software, Inc. All rights reserved.

For original and rewritten copy see:

<http://c2.com/cgi-bin/wiki?WindowPerTask><http://c2.com/ppr/ui.html>**[Beck, 95- 2]****Beck, Kent .*****Dividing Window into a Few Panes.***

Copyright 1995, Fiest Class Software, Inc. All rights reserved.

For original and rewritten copy see:

<http://c2.com/cgi-bin/wiki?WindowPerTask><http://c2.com/ppr/ui.html>

[Bennett, 95]

Bennett, Ian.

HALI The International Magazine of Antique Carpet and Textile Art.

volume 74 (1995), pp. 87-95

<http://www.thunderstrand.com/rugmags.html>

[Berkun, 99]

Berkun, Scott.

(Training manager for UI design and usability at Microsoft)

Making Usable Products: An Informal Development Process for Good UI.

Human Factor: Bi-monthly Column on UI Design, March 1999.

http://msdn.microsoft.com/library/default.asp?URL=/library/welcome/dsmsdn/hfac/tor8_2.htm

[Bjerknes, 87]

Bjerknes, G., Ehn, P., and Kyng, M.

Computer and Democracy: A Scandinavian Challenge.

Aldershot, Avebury, Great Britain.

[Blank, 98]

Blank, Peter.

Peter Eisenman.

Stanford Preseidential Lectures and Symposia in the Humanities and Arts,

Stanford University, 1998

<http://prelectur.stanford.edu/lecturers/eisenman/index.html>

[Bowker, et al., 97]

Bowker, Geoffrey C. Star, Susan Leigh, (University of Illinois, Urbana-Champaign) Turner, William (CERESI/CNRS) Gasser, Les (University of Southern California).

SOCIAL SCIENCE, TECHNICAL SYSTEMS AND COOPERATIVE WORK

Beyond the Great Divide.

A Volume in the Computers, Cognition and Work Series.

Lawrence Erlbaum Assoc, 1997.

[Brown, 97]

Brown, M.

An Introductory Course on C.

Department of Electronics and Computer Science
University of Southampton
<http://ftp.uni-klu.ac.at/programming/c/Cpp/notes.html>

[Burks, 95]

Burks, Larry D.

A Structuralist Approach to Electronic Media.

Harvard Graduate School of Design

Last updated: 2 October 1995

http://www.gsd.harvard.edu/~lburks/f1/research/fall95/str_media.html

[Buschmann, et al., 96]

Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter;
Stal, Michael.

Pattern Oriented Software Architecture: A System of Patterns.

John Wiley & Sons, 1996

This book covers more or less the same ground as the Gang of Four book [Gamma, et al., 95] but it covers some additional material as well. In particular, the Model/View/Controller and Presentation/Abstraction/Control architectures are discussed in depth:

<http://www1.fatbrain.com/asp/bookinfo/bookinfo.asp?theisbn=0471958697&from=NCN454>

[Carmel, et al., 93]

Carmel, E., Whitaker, R. D., and Geoge, J. F.

PD and Joint Application Design: A transatlantic Comparison.

Communications of ACM, Vol. 36, No. 4, June 1993.

[Catarci et. al., 96]

Catarci, T.; Cruz, I. F.

Special Issue on Information Visualization.

ACM Sigmod Record 25(4), 1996.

[Chai, 2000*]

Chai, Ian.

Visualesson Patterns.

<http://st-www.cs.uiuc.edu/~chai/writing/visualessons/index.html>

[Comp.object, 97]

comp.object.

Change an Object's Class at Run-Time (State Pattern).

comp.object, 97.

<http://ootips.org/state-pattern.html>

[Coplien, et al., 95]

Coplien, James O.; Schmidt, Douglas C.

Pattern Languages of Program Design 1.

Addison-Wesley Pub Co; 1995

[Coplien, 96]

Coplien, James O.

Software Patterns.

SIGS Books, New York, New York, 1996

[Coplien, 97-1]

Coplien, James O.

Idioms and Patterns as Architectural Literature .

IEEE Software Special Issue on Objects, Patterns, and Architectures, January, 1997

[Coplien, 97-2]

Coplien, James O.

On the Nature of The Nature of Order.

The Chicago Patterns Group;

Tuesday, 5 August 1997

[notes taken by Brad Appleton]

<http://www.enteract.com/~bradapp/docs/NoNoO.html>.

[Coplein, 98-1]

Coplien, James O.

Space: The Final Frontier.

C++ Report 10(3), March 1998, pp. 11 - 17

<http://www.bell-labs.com/~cope/Patterns/C++Report/SpaceFinalFrontier-1.html>

[Coplein, 98-2]

Coplien, James O.

Geometry of C++ Objects.

C++ Report 10(9), October 1998, pp. 40 – 44
<http://www.bell-labs.com/~cope/Patterns/C++Report/IdiomGeometry/IdiomGeometry-1.html>

[Coplien, 98-3]
 Coplien, James, O.
Writers Workshop Patterns
 Last edited November 21, 1998
<http://c2.com/cgi/wiki?WritersWorkshopPatterns>

[Coplein, 99-1*]
 Coplien, James O.
A Pattern Definition: Software Patterns.
<http://hillside.net/patterns/definition.html>

[Coplein, 99-2*]
 Coplien, James O.
Software Design Patterns: Common Questions and Answers.
<ftp://st.cs.uiuc.edu/pub/patterns/papers/PatQandA.ps>

[Coplein, 99-3]
 Coplien, James O.
Multi-Paradigm Design for C++
 Addison-Wesley, 1999.

[Cunningham, 99*]
 Cunningham, Ward.
History Of Patterns
 WikiWikiWeb
 Last edited August 27, 1999
<http://c2.com/cgi-bin/wiki?HistoryOfPatterns>

[Damian, et al., 99]
 Damian, Adrian. Hong, Danfeng. Li, Holly. Pan, Dong.
Joint Application Development and Participatory Design.
 Deptment of Computer Science, University of Calgary
<http://www.cpsc.ucalgary.ca/~pand/seng/613/report.html>

[Dart, 94]

Dart Susan A.

Adopting An Automated Configuration Management Solution.

Dart Technology Strategies, Inc.

Conference: STC'94 (Software Technology Center), 12 April, 1994, Utah

<http://www.cwi.com/developers/developersACEE.html#20>

[Detweiler, et al., 96]

Detweiler, Mark C. (Penn State University) Omanson, Richard C. (Ameritech)

Ameritech Web Page User Interface Standards and Design Guidelines.

© Copyright Ameritech Corporation, 1996

http://www.ameritech.com/corporate/testtown/library/standard/web_guidelines/index.html

[Deugo, 98]

Deugo, Dwight.

Pattern Introduction.

<http://scs.carleton.ca/~deugo/95590/Notes/introduction/introduction.html>

[DISE, 98]

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING,

Virginia Polytechnic Institute and State University.

Macroergonomics and Sociotechnical Systems Theory .

<http://pelican.ise.vt.edu/model/index.html>

(Select: Macroergonomics and Sociotechnical System Theory)

For recent Socio-Technical view points see **[Bowker, 97]**

[DP2, 94]

Doors of Perception 2 @HOME Conference.

Who is who @ Mediamatic - Christopher Alexander.

4-6 November 1994, Amsterdam

<http://www.mediamatic.nl/whoiswho/alexander/index.html>

[Duell, 97]

Duell, Michael.

Non-software examples of software design patterns.

Object Magazine, Jul 97.

[Encyclopedia, 2000]

<http://www.encyclopedia.com/>

[EuroPLOP, 97]

EuroPLOP 97

Second European Conference on Pattern Languages of Programming.

July 10 - 12, 1997, Kloster Irsee, 87660 Irsee, Germany

<http://siesta.cs.wustl.edu/~schmidt/europlop-97/workshops.html>

[EuroPLOP, 98]

EuroPLOP 98

Third European Conference on Pattern Languages of Programming.

July 9 - 11, 1998, Kloster Irsee, 87660 Irsee, Germany

<http://www.coldewey.com/europlop98/>

[EuroPLOP, 99]

EuroPLOP 99

Forth European Conference on Pattern Languages of Programming.

July 8 - 10, 1999, Kloster Irsee, 87660 Irsee, Germany

<http://www.argo.be/europlop/writers.htm>

[Evans, 96]

Evans, Roger.

Book that Changed my Life.

The Quarterly Journal of the Urban Design Group;

Issue 59, July 1996;

http://rudi.herts.ac.uk: 80/ej/udg/59/b_evans.html

[Feenberg, 91]

Feenberg, Andrew.

Critical Theory of Technology.

Oxford University Press, 1991.

[Foley et al , 95]

Foley, Jim. and Pitkow, James.

Research Priorities For The World-Wide Web.

October 31, 1994.

<http://www.cc.gatech.edu/gvu/nsf-ws/report/Report.html>

[Foots, 91]

Foots, Brian.

A Fractal Model of the Lifecycle of Reusable Objects.

(extended abstract (1))

OOPSLA '91 Workshop on Reuse

Ottawa, Ontario, Canada

<http://www.laputan.org/frameworks/fractal.html#one>

[Fowler, 96]

Fowler, Martin.

Analysis Patterns : Reusable Object Models.

Addison-Wesley Object Technology Series

[Fultz, 99*]

Fultz, Jenny.

DeStijl and Bauhaus

<http://users.anderson.edu/~jfultz/destijl.html>

[Gabriel, 96]

Gabriel, Richard P.

Patterns of Software; Tales from the Software Community.

New York: Oxford University Press, 1996;

with a foreword by Christopher Alexander.

[Gabriel, 99*]

Gabriel, Richard P.

A Pattern Definition.

(Patterns Home Page)

<http://hillside.net/patterns/definition.html>

[Gamma, et al., 95]

Gamma, Erich. Helm, Richard. Johnson, Ralph. Vlissides, John.

Design Patterns: Elements of Reusable Object-Oriented Software.

Addison-Wesley, 1995.

<http://www1.fatbrain.com/asp/bookinfo/bookinfo.asp?theisbn=0201633612&from=NCN454>

[Gosling , 96-1]

Gosling, James. Yellin, Frank.

The Java Team

Java API Documentation

Version 1.0.2

<http://java.sun.com/products/jdk/1.0.2/api/java.awt.Window.html#22917>

[Gosling, 96-2]

Gosling, James. Yellin, Frank.

The Java Team

Java API Documentation

Version 1.0.2

<http://java.sun.com/products/jdk/1.0.2/api/java.awt.Panel.html>

[Grabar, 92]

Grabar, Oleg.

The Mediation of Ornament.

Princeton University Press, 1992.

[Gray, et. al., 93]

Gray, W. D., John, B. E., and Atwood, M. E.

Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance.

Human-Computer Interaction, 8, 1993, pp. 237-309.

[Harris, et al., 96]

Harris, G. B., and Taylor S.

Participatory Design Using the Action Workflow Model,

Workshop on Strategies for Collaborative Modeling and Simulation,

Conference on Computer-Supported Cooperative Work, Boston, MA.

[Harrison, et al., 99]

Harrison, Neil., Foote, Brian., Rohnert, Hans.

Pattern Languages of Program Design 4 (Software Patterns Series)

Addison-Wesley Pub Co; 1999

[Heather, 98]

Duggan, Heather.

Virtual Community Consultant

Timeless Way of Building.

Summary of a book by Christopher Alexander.

Last update: 1/18/98

http://www.bigbangworkshops.com/html/timeless_way.html

[Heiniger, 99]

Heiniger, Daniel

Model View Controller.

last edited November 8, 1999

<http://c2.com/cgi-bin/wiki?ModelViewController>

[Holub, 99]

Holub, Allen.

Build user interfaces for object-oriented systems, Part 2: The visual-proxy architecture

A scalable architecture for building object-oriented user interfaces

<http://www.javaworld.com/javaworld/jw-09-1999/jw-09-toolbox.html>

[HRM, 99-1]

Halifax Regional Municipality

Road Network - Hierarchy of Roads - Issues and Opportunities.

Street classification system

<http://www.region.halifax.ns.ca/Regops/Transerv/Traffic/Work36.html#road>

[HRM, 99-2]

Halifax Regional Municipality

The roles of Transit and Traffic Demand Management.

Street classification system

<http://www.region.halifax.ns.ca/Regops/Transerv/Traffic/Work36.html#road>

[Huston, 97]

Huston, Vince

A short description of the State pattern

<http://rampages.onramp.net/~huston/dp/state.html>

[IBM, 2000]

IBM

Ease of Use

http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/558

[infoplease, 99-1*]

infoplease.com

Dictionary

<http://www.infoplease.com/ipd/A0580517.html>

[infoplease, 99-2*]

infoplease.com

Dictionary

<http://www.infoplease.com/ipd/A0580540.html>

[infoplease, 99- 3*]

infoplease.com

Dictionary

<http://www.infoplease.com/ipd/A0490809..html>

[Jencks, 97]

Jencks, Charles.

The Architecture of the Jumping Universe : A Polemic : How Complexity Science Is Changing Architecture and Culture

John Wiley & Son Ltd., 1997

[Karat, 90]

Karat, C. M.

Cost-Benefit Analysis of Usability Engineering Techniques,
 Proceedings of the Human Factors Society 34th Annual Meeting,
 Volume 2, Orlando, FLA, Oct 1990, pp. 839-843.

[Khare, 95]

Khare, Rohit.

On the diffusion of Christopher Alexander's A Pattern Language into Software Architecture

<http://xent.ics.uci.edu/~khare/Alexander.html#/>

[Ladkin, 96]

Ladkin, P.

Inside Risks Forum,

<http://csrc.nsl.nist.gov/rskforum/risks18.010.1996>.

[Laurel, 93]

Laurel, Brenda.
Computers As Theater.
 Reading, MA: Addison-Wesley, ©1993.

[LC, 99*]
Library of Congress
<http://www.loc.gov/>

[Lea, 93]
Lea, Doug.
Christopher Alexander: An Introduction for Object-Oriented Designers
<http://g.oswego.edu/dl/ca/ca/ca.html>

[Lee, 92]
Lee, Allen S.
Architecture as a Reference Discipline for MIS.
 Working Paper, University of Minnesota, Minneapolis, Minnesota

[Lemon, et al., 99]
Lemon Bill, Pynadath David, Taylor Glenn, Wray Bob.
Cognitive Architectures, A hypertext analysis of Architectures for Intelligence.
 Advanced Technology Laboratories, University of Michigan.
<http://krusty.eecs.umich.edu/>

[Leveson, et al., 93]
Leveson, N. G. and Turner, C. S.
An Investigation of the Therac-25 Accidents,
 IEEE Computer, Jul 1993, 26(7), pp. 18-41.

[Lynn, 88]
Lynn, Arthur Steen,
The Science of Patterns.
 Science, 240 (1988): 611-616.

[Mantei, et al., 88]
Mantei M. M. and Teorey, T. J.
Cost/Benefit Analysis for Incorporating Human Factors in the Software Lifecycle,
 CACM, Apr 1988, 31(4), pp. 428-439.

[Martin, et al., 97]

Martin, Robert C., Riehle, Dirk., Buschmann, Frank., Vlissides, John.
Pattern Languages of Program Design 3 (Software Patterns Series)
 Addison-Wesley Pub Co; 1997

[Mascha & Seethaler, 96]

Mascha & Seethaler,
Relaxation of controlling mechanisms by deregulation.
 Millennium workshop 96-Vienna.
<http://www.architects.co.at/mod02/mod02-02.html>

Also, see the original source (in German):

<http://www.architects.co.at/>

[Meszaros, 95]

Meszaros, Gerard.
Pattern: Half-Object + Protocol
<http://www.belllabs.com/topic/books/PLoPD1/manuscripts/HalfObjectPlusProtocol.html>

[Meszaros et al., 97]

Meszaros, Gerard & Doble, Jim.
A Pattern Language for Pattern Writing;
http://hillside.net/patterns/Writing/pattern_index.html

[Microsoft, 2000]

Microsoft: The Windows User Experience and Interface Design team
User Experience and Interface Design Resources
<http://msdn.microsoft.com/UI/default.asp>

[Miller, 93]

Miller, E. S.
From System Design to Democracy,
 Communications of ACM, Vol. 36, No. 4, June 1993.

[Milne, 96]

Milne, Bruce T.
Self-Organizing Systems.

Landscape Ecology & Macroscopic Dynamics.
 University of New Mexico, USA, Biology 576
<http://algodones.unm.edu/~bmilne/bio576/instr/html/SOS/sos.html>

[Murdocca, et al., 99]
 Murdocca, Miles J. and Heuring, Vincent P.
Principles of Computer Architecture.
 Chapter1: introduction.
 © Copyright 1999 Prentice Hall.
<http://www.cs.rutgers.edu/~murdocca/POCA/>

[Myers, et al., 92]
 Myers, B. A. and Rosson, M. B.
Survey on User Interface Programming,
 Proceedings SIGCHI'92: Human Factors in Computing Systems.
 Monterrey, CA, May 3-7, 1992. 195-202.

[Myers, 96]
 Myers, B. A.
A Quick History of Human Computer Interaction.
 Carnegie Mellon University; School of Computer Science; Technical Report.
 CMU-CS-96-163 and
 Human Computer Interaction Institute; Technical Report
 CMU-HCI-96-103, August 1996.

[Myers, et al., 96]
 Myers, Brad. Carnegie Mellon University; Hollan, Jim. University of New Mexico; Cruz, Isabel. Tufts University; (Editors).
Strategic Directions in Human Computer Interaction
 Revised; Final version of November 13, 1996
<http://www.cs.cmu.edu/~bam/nsfworkshop/hcireport.html>

This article is prepared based on the results of the Human Computer Interaction Working Group of the ACM Workshop on Strategic Directions in Computing Research, and was authored by:
 Steve Bryson, NASA Ames Research Center, Dick Bulterman, CWI, Tiziana Catarci, University of Rome, Wayne Citrin, University of Colorado Boulder, Isabel Cruz, Tufts University (co-chair and editor), Ephraim Glinert, RPI, Jonathan Grudin, University of California Irvine, Jim Hollan, University of New Mexico (editor), Yannis Ioannidis, University of Wisconsin-Madison, Rob Jacob, Tufts

University, Bonnie John, Carnegie Mellon University, David Kurlander, Microsoft Research, Brad Myers, Carnegie Mellon University (co-chair and editor), Dan Olsen, Carnegie Mellon University, Randy Pausch, University of Virginia, Stuart Shieber, Harvard University, Ben Shneiderman, University of Maryland College Park, John Stasko, Georgia Tech, Gary Strong, NSF, Kent Wittenburg, Bellcore.

Note:

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM Inc., fax +1 (212) 869-0481, or permissions@acm.org.

[Nasr,73]

Nasr, Seyyed Hossein.

Foreword on: **[Ardalan, 73]**

[Neumann, 91]

Neumann, P. G.

Inside Risks: Putting on Your Best Interface,

CACM, Mar 1991, 34(3), p. 138.

[Nielsen, et. al., 93]

Nielsen, J. and Landauer, T. K.

A Mathematical Model of the Finding of Usability Problems,

Proceedings INTERCHI'93: Human Factors in Computing Systems, Amsterdam, The Netherlands, Apr, 1993, pp. 206-213.

[OGC, 99]

Oxford General Catalogue

Last updated Saturday, 04-Dec-99 09:09:48

<http://199.72.49.25/docs/0195037537.html>

[Parnas, 78]

Parnas, David L.

Designing Software for Ease of Extension and Contraction. Proc.
3rd Int. Conf. Soft. Eng., May 1978.

[Perkins, 97]

Perkins, Graham.

The State Pattern in Smalltalk

(A long description of the State pattern)

<http://www.mk.dmu.ac.uk/~gperkins/Smalltalk/State/>

[PLOP, 96]

PLOP 96

Allerton Park, Illinois, USA, Sept. 4-6, 1996

<http://www.cs.wustl.edu/~schmidt/PLoP-96/program.html>

[PLOP, 97]

PLOP 97

September 3-5, 1997; Allerton Park; Monticello, Illinois, USA

<http://st-www.cs.uiuc.edu/users/hanmer/PLoP-97/Workshops.html>

[PLOP, 98]

PLOP 98

August 11-14, 1998; Allerton Park; Monticello, Illinois, USA

<http://st-www.cs.uiuc.edu/~plop/plop98/>

[PLOP, 99]

PLOP 99

August 11-14, 1999; Allerton Park; Monticello, Illinois, USA

<http://st-www.cs.uiuc.edu/~plop/plop99/proceedings/>

[Riehle, et al., 96]

Riehle, Dirk and Zullighoven, Heinz.

Understanding and Using Patterns in Software Development.

UBILAB, Union Bank of Switzerland / University of Hamburg, Germany.

http://www.ubs.com/e/index/about/ubilab/ext/publications/e_rie96c.htm

[Rohrer, 95]

Rohrer Tim

Feelings Stuck in a GUI web: metaphors, image-schemata, and designing the human computer interface

Center for the Cognitive Science of Metaphor Online

University of Oregon Philosophy Department.
<http://darkwing.uoregon.edu/~rohrer/gui4web.htm>

[Rudofsky, 64]
Rudofsky, Bernard.
Architecture without Architect.
 Academi Edition, London, 1964.

[Salingaros, 95]
Salingaros, Nikos A
In Defense of Alexander
 HALL: The International Magazine of Antique Carpet and Textile Art, volume 78
 (1995), pages 67-69.
<http://sphere.math.utsa.edu/sphere/salingar/DefenseAlex.html>

[Salingaros, 97-1]
Salingaros, Nikos A.
Some Notes on Christopher Alexander.
<http://www.math.utsa.edu/sphere/salingar/Chris.text.html>

[Salingaros, 97-2]
Salingaros, Nikos A.
Some Notes on Christopher Alexander.
<http://www.math.utsa.edu/ftp/salingar/NatureofOrder.html>

[Salingaros, 99]
Salingaros, Nikos A.
Architecture, Patterns, and Mathematics
 Division of Mathematics; University of Texas at San Antonio,
http://www.leonet.it/culture/nexus/network_journal/Salingaros.html

[Salo, 94]
Salo, Heikki.
Hyperboarding: From Aristotle to Brenda Laurel and on...
<http://www.cs.uta.fi/~hes/AtoBL/index.html>

This essay was originally delivered as a three hour lecture in a Nordic seminar on novel storyboarding techniques. The seminar was organized by the Finnish Museum of Modern Art and Nordic Council in liaison with the ISEA 94 symposium in Helsinki.

[Saunders, 97]

Saunders, Robert.

Pattern Language by Christopher Alexander

Last modified: Fri Jun 20 16:25:12 EST 1997

<http://www.arch.su.edu.au/~rob/study/PatternLanguage.html>

[Schumacher, 95]

Schumacher, Jr. Robert M.

Ameritech Graphical User interface Standards and Design Guidelines

© Copyright Ameritech Corporation, 1992-1995.

<http://www.ameritech.com/corporate/testtown/library/standard/std-guix.html>

[Schwartz, et al. , 95]

Schwartz, Amy L. and Hardzinski Mary L.

Ameritech Phone-based User Interface Standards and Design Guidelines

© Copyright Ameritech Corporation, 1992-1995.

<http://www.ameritech.com/corporate/testtown/library/standard/std-pbix.html>

[Scully,74]

Scully, Vincent Joseph.

Modern Architecture: The Architecture of Democracy

Taschenbuch - 158 Seiten (Februar 1974)

George Braziller Inc.

[Sharp, 91]

Sharp, Dennis.

The Illustrated Encyclopedia of Architects and Architecture.

New York: Quatro Publishing, 1991

[Sibert et al. , 93]

Sibert, John. and Marchionini, Gary.

Human-Computer Interaction Research Agendas.

Behavior and Information Technology, 12(2), Mar-Apr, 1993.pp. 67-135.

[Spacesyntax, 99]

Space Syntax laboratory, University College London.

Stockton-on-Tees; Movement from Attractor and Pedestrian movement in the Town Center

[Http://www.bartlett.ucl.ac.uk/spacesyntax/pedestrian/pedestrian.html](http://www.bartlett.ucl.ac.uk/spacesyntax/pedestrian/pedestrian.html)

[Stingray, 99]

stingray.com.

The Model-View-Controller (MVC) Framework

(Stingray is a division of Rogue Wave Software)

<http://support.stingray.com/mvc.asp>

[Strong, 93]

Strong, G.W.

New Directions in Human-Computer Interaction Education, Research and Practice.

NSF Workshop. Sept. 11, 1994.

<http://www.sei.cmu.edu/arpa/hci/directions/TitlePage.html>

[Tayfun, 97]

Tayfun Kalyoncu

All about Turkish carpets & rugs

<http://www.about-turkey.com/carpet.htm>

[Tidwell, 98]

Tidwell, Jenifer.

A Pattern Language for Human-Computer Interface Design

http://www.mit.edu/~jtidwell/interaction_patterns.html

[Tunstall, 99]

Tunstall, Tom.

Re: outsourcing

<http://www.outsourcing-experts.com/cgi-bin/webbbs/outsourcing-board/index.cgi?read=1749>

[UWM, 97]

The University of Wisconsin-Madison Writing Center

Writer's Handbook,

Documentation : Chicago Style.

<http://www.wisc.edu/writing/Handbook/DocChicago.html>

[UNESCO, 97]

**UNESCO - ENCYCLOPEDIA OF LIFE SUPPORT SYSTEMS (EOLSS) -
RESEARCH PROJECT**

**1.29. Knowledge Management, Organizational Intelligence and Learning,
and Complexity**

<http://www.utdallas.edu/dept/socsci/Fourth.htm>

[UOPO, 92-1]

University of Oregon Planning Office

University of Oregon's Campus Planning Processes

(commonly referred to as the Oregon Experiment)

October 1992

<http://darkwing.uoregon.edu/~uplan/planexp.html>

[UOPO, 92-2]

University of Oregon Planning Office

http://geography.uoregon.edu/infographics/campus_maps.html

[Vlissides, et al., 96]

Vlissides, John M., Coplien, James O., Kerth, Norman.

Pattern Languages of Program Design 2.

Addison-Wesley Pub Co; 1996.

[Watanabe, et al., 92]

Watanabe, Osamu. et al.

Kolmogorov Complexity and Computational Complexity.

Springer_Verlag, 1992.

[Webb, 95]

Webb, Jayson M.

Ameritech Character-based Interface Standards and Design Guidelines

@ Copyright Ameritech Corporation, 1992-1995.

<http://www.ameritech.com/corporate/testtown/library/standard/std-text.html>

[WebCat, 98]

WebCat: E&S / Hunt/Mellon Libraries

(Keyword: Christopher Alexander)

<http://webcat.library.cmu.edu/uhtbin/cgiisirs/RWy3tbVKub/10294199/60/30776>

[White, 99]

White, Bebo.

Using Hypertext Methods in Web Authoring (WDE)

Stanford Linear Accelerator Center (SLAC)

<http://www.slac.stanford.edu/~bebo/techapps/sld001.htm>

[Willoughby, 93]

Willoughby, William T.

FOUNDING NIETZSCHE IN THE FIN D'OU T HOU S

ARCHITRONIC The Electronic Journal of Architecture Volum2, December 1993

<http://arcrs4.saed.kent.edu/Architronic/v2n3/v2n3.05.html>

[Wolfe, 81]

Wolfe, Tom.

From Bauhaus to Our House

Farrar, Straus and Giroux 1981

<http://www.tomwolfe.com/bauhaus.htm>

A Proto Pattern Language for Human-Computer Interaction

Glossary

Dead End

A node that does not have any path to any other node except to its main node is a dead end. The concept of dead ends is the same as dead end for the network of the streets of a town.

Entry Node

The first node that you can navigate when you open an informative artifact is the entry node. The same as the concept of entry in a building.

Informative Artifact

"An artifact that is giving information, is an informative artifact. "

Example: " an informative book."

For more information about "Informative" as a term, see:

<http://www.infoplease.com/ipd/A0490809.html>

Informative Space

Informative space is the space of an informative artifact

Interaction Domain

The domain that negotiates between human factor and technology.

For more information see: Figure1 and its description in Chapter1

(Morphology of the Concept) of this thesis.

Metaphor

Metaphor is a figure of speech in which a word or phrase denoting one kind of object or action is used in place of another to suggest a likeness or analogy between them. (See also modeling)

For more information about metaphorical view to the space of an informative artifact see:

Rohrer, Tim.

Feelings Stuck in a GUI web: metaphors, image-schemata, and designing the human computer interface.

Center for the Cognitive Science of Metaphor Online

University of Oregon Philosophy Department.

<http://metaphor.uoregon.edu/gui4web.html>

Miniature

A miniature does reflect only those inherited information's from its model that can be represented in its scale. It is not a copy of the model. The same concept can be seen in the relation between thread and process.

Thread is a miniature of the process.

Modeling

Modeling is an effort to abstract the basic principle of one phenomenon to

be used as the principle for creating another phenomena.

Node

A node is an informative object. It is an entity that has informative state and functionality. This concept is borrowed from object-oriented paradigms.

For a definition of Object in Object Oriented paradigms see:

<http://www.uow.edu.au/~hasan/buss317/ooterms.htm>

Transparency

If an interface reflects the principle of its structure (background system) then it is transparent.

For a definition of transparency in the scope of this thesis see section 5 of chapter 4 (Background: Forces)