

MATHEMATICAL STRUCTURES IN PATTERN ORGANIZATIONS

SALUKA R. KODITUWAKKU

Department of Statistics and Computer Science, University of Peradeniya, Sri Lanka

E-mail: salukak@pdn.ac.lk

ABSTRACT

Since the introduction of design patterns, a large number of patterns have been identified and documented. As a result, patterns in the literature relate one another in different ways. Unfortunately, most of these patterns are not properly organized. In applying these patterns in to problems at hand novice designers encounter many difficulties such as identification and selection of collectively applicable set of patterns that suits the problem at hand. We investigate mathematical structures in existing pattern organization techniques such as pattern catalogues, pattern systems and pattern languages. Then we attempted to use these mathematical structures in developing a new organization technique. This paper proposes a new organization method and illustrates it with a collection of object oriented patterns drawn from the literature. Our investigation indicated that existing pattern organizations form graph structures and categorical structures. These structures are formed by patterns and relationships among them. The proposed organization method organizes patterns into sequences according to the relationships among patterns. This organization consists of two types of categories: major category and alternative category. The major categories are defined based on the Uses relationship and the alternative categories are defined based on the Variants, Refines and Conflicts relationships. Each major category consists of patterns that have potential to form a pattern system or a pattern language. So they assist practitioners in finding a sequence of related patterns that can be collectively applied to solve complex problems. Each alternative category consists of patterns that provide alternative solutions to the same problem or similar problems. It assists in finding a number of possible solutions (patterns) to a particular problem. The proposed organization combines these two types of categories by structuring them into pattern

sequences. A pattern sequence consists of a major category and a collection of related alternative categories. Pattern sequences assist practitioners in finding a collectively applicable patterns and possible alternatives to them. So pattern sequences facilitate the selection of most appropriate sequences of patterns to solve complex problems without searching through the pattern literature.

Key Words: Design Patterns, Pattern Catalogues, Pattern Systems, Pattern Languages, Pattern Sequences, Category Theory, Graphs

INTRODUCTION

Although work on software patterns started in late 1980s, it was not until the early 1990s that patterns began to attract more attention in software industries. Gamma *et al.* (1994) have documented some successful designs as object oriented design patterns. In the last ten years or so, software patterns have grown exponentially and become very popular in the software industry. The goal of the pattern community is to create a communication language to share knowledge and experience of software design.

With ever-growing interest of patterns, the pattern community has identified and documented a large number of patterns. However, there is no unified method for describing and organizing these patterns. Some of the existing patterns are organized into *Pattern Catalogues*, *Pattern Systems* or *Pattern Languages*. Except for the pattern languages, pattern Catalogues and pattern systems hardly address the relationships between patterns. When applying a pattern to a particular problem, a user must first find the most suitable one. Experienced pattern users are able to find and select the most suitable pattern to their problem, whereas, novice pattern users would need some guidelines before deciding which pattern to be applied.

Patterns are rooted in many disciplines, including Physics, Biology and Chemistry. The work on software patterns is largely influenced by Alexander's work on urban planning and building construction. Alexander describes the philosophic aspect of the patterns in "*The Timeless Way of Building*" (Alexander, 1979), and their practical aspect in "*A Pattern Language*" (Alexander, 1977). According to Alexander, "every pattern we define must be formulated in the form of a rule, which establishes a relationship between context, a system of forces which arises in that context, and a

configuration, which allows these forces to resolve themselves in that context (Alexander, 1979)".

Alexander (1979) organized his patterns into "A Pattern Language". Patterns in this language are arranged from large-scale to small-scale and are connected. Each pattern is connected to other patterns in two ways. First, it has an opening context, which explains how the pattern helps to complete certain large patterns. Second it ties up with all the smaller patterns that help to complete or refine this pattern, which we call the resultant context or closing context. Both the resulting and opening contexts help the user decide which pattern should be applied next. In contrast, most software patterns exist in isolation and a user must make his own decision as to which pattern is to be used next. In this paper, we describe mathematical structures in existing pattern organizations and then organize individual patterns into useful groups according to such mathematical structures.

MATERIALS AND METHODS

First, we discuss existing pattern organization techniques and relationships among patterns. Then we analyze these organizations and derive mathematical structures formed by patterns and relationships among them. Finally, we describe how individual patterns can be organized according to such mathematical structures in order to assist pattern users. The proposed method is illustrated with object oriented design patterns derived from the literature.

Pattern Organization Techniques

"A design pattern is an abstraction from a concrete recurring solution that solves a problem in a certain context" (Gamma *et al.*, 1994). Typically, a software design pattern has a Name, a Context describing applicability of the pattern, a Problem addressing the constraints which make hard to solve the problem, a Solution describing how forces are resolved, Related Patterns referring to the other patterns, and Consequences describing benefits and drawbacks of application of the patterns. Authors of object oriented design patterns incorporated additional components such as Participants describing the components of the solution structure, Collaborations explaining interactions among participants and Implementation Guidelines including program fragments.

Design patterns were popularized after the publication of *Design Pattern* (Gamma *et al.*, 1994) by a group known as *Gang-Of Four (GOF)*. They introduced a pattern catalogue consisting 23 patterns. These patterns are organized according to the purpose of patterns. All these patterns address common design problems in Object-Oriented design. Even though it provides a collection of patterns it provides little assistance in selecting patterns. The *System of Patterns* book (Buschmann, 1996) also introduced a collection of pattern systems for object oriented design. These are large in scale compared to the patterns in *Gang-Of Four* catalogue. Later on several pattern collections (Coplien 1995; Riehle *et al.*, 1998), were introduced by the pattern community.

Pattern Catalogues

Individual patterns, which address a single design problem, are organized into pattern Catalogues. The GOF (Gamma *et al.*, 1994) organized them as creational patterns, structural patterns and behavioral patterns based on the pattern's scope. Patterns of software architecture (Buschmann, 1996) also organize patterns as architectural patterns, design patterns and programming pattern based on the scale of patterns. Individual patterns are also Catalogues based on the purpose of the pattern. The patterns in a catalog indicate other patterns to consider but do not explicitly specify the relationships between patterns. So, pattern Catalogues leave the user unable to understand the patterns and relationships between patterns. In effect, novice users may not able select the most appropriate or collection of patterns to a problem at hand. Pattern catalogues provide little assistance in selecting a set of related patterns to solve complex problems.

Pattern Systems

A pattern system is a set of related patterns, which work together to support the construction and evolution of whole architectures. Pattern systems are organized into related groups and subgroups at multiple levels of granularity, describe the many interrelationships between the patterns and their groupings and describe how they may be combined and composed to solve more complex problems. Pattern systems are created from the individual patterns within Catalogues. For example, Model-view-Controller (Buschmann, 1996) is created from the Observer, Strategy, and Composite patterns, which are members of the *GOF's* catalog. Even though pattern systems

provide large scale solutions and combine related individual patterns together, there are a limited number of systems available. Additionally researchers put more emphasis on documenting new individual patterns rather than combining individual patterns into systems or any other useful collections.

Pattern Languages

A pattern language is a collection of related patterns in which each of its patterns collaborates to solve complex problems that are not explicitly addressed by any individual pattern. A pattern language includes rules and guidelines, which explain how/when to apply its patterns to solve a problem, which is larger than any individual pattern, can solve. These rules and guidelines suggest the order and granularity for applying each pattern in the language. Pattern languages documented targeting different application domains can be found from (Coplien, 1995; Riehle *et al.*, 1998; John *et al.*, 1996; Alexander, 1977). Pattern languages assist users in selecting complete collection of related patterns to solve complex problems.

Alexander introduced the theory of patterns and he used them to document a set of patterns, *A Pattern Language* (Alexander, 1977), for urban planning and building construction. Alexander's pattern language is a complete language as it describes the recurring problems in the entire domain and it is organized based on the *Uses* relationship. Software pattern languages are also organized based on the *Uses* relationship. However, software pattern languages are complete within a particular sub-domain. Since pattern languages evolved from individual patterns, it is important to organize individual patterns that are related by the *Uses* relationship into languages or sequences.

Relationships between Patterns

Since patterns do not exist in isolation, each pattern relates to other existing patterns. In Alexander's pattern language, large-scale patterns contain small-scale patterns and a small-scale pattern is embedded within a number of large-scale patterns. Therefore composition relationship can be explicitly identified from Alexander's patterns.

With ever-growing interest in software patterns, patterns in the literature relate to one another in a variety of ways. Walter Zimmer classified three types of relationships between design patterns: X uses Y in its solution, Variant of X uses Y in

its solution, and X is similar to Y (Zimmer, 1995). James Nobel identified the three primary relationships between Object-oriented design patterns: Uses, Refines and Conflicts. He also proposed a set of secondary relationships: Used By, Refined By, Variants, Variant Users, Similarity, Combines, Requires, Tiling and Sequence of Elaboration (Noble, 1998). In this paper, we have considered four common relationships; Uses, Specialization, Alternative and Variants.

Uses

One pattern uses another pattern when the second problem solves a sub-problem raised by the application of the first pattern. For example, the Model-View-Controller pattern uses *Strategy*, *Composite* and *Observer* patterns (Buschmann, 1996; Gamma *et al.*, 1994).

Alternatives

A set of patterns may propose mutually exclusive solutions to similar problems. In *Design Pattern*, creational patterns provide alternative solutions for object creation. For example, *Decorator* and *Strategy* patterns (Gamma *et al.*, 1994) provide solutions for modifying the behavior of other objects. In Alexander's language, a number of alternative patterns can be used as *Subculture Boundary* (Alexander, 1977).

Specialization

One pattern deals with a specialization of the problem, the same set of forces as another pattern addresses, but may address additional forces, and has a similar but more specific solution structure. For example, Object-Lifetime-Manager pattern (Levine *et al.*, 1999) specializes the Singleton pattern (Gamma *et al.*, 1994).

Variants

Some kinds of design problems and solutions occur more frequently than the other problems and solutions. Therefore, the method of instantiating patterns is more common in practice than other methods. Pattern authors address these variant situations by providing alternative solutions for the same problem or providing a single solution to a number of different problems. James Nobel classifies such patterns as pattern *Variants* and decomposes such variants into *Solution Variants* and *Problem Variants* (Noble, 1998). In *Design Pattern*, *Adapter* pattern provides two solutions,

Class Adaptor and *Object Adapter*, to the same problem. *Proxy* pattern provides a surrogate for another object and there are several variants of *Proxy* such as *Remote Proxy*, *Virtual Proxy*, *Access Control Proxy* and so on. Each of these *Proxy* patterns can be used to solve different problems, which require surrogates.

Although only a single example was given for each relationship many such examples can be found from the literature. Identification of related patterns paves the way for organizing patterns into useful collections that assist practitioners in selecting patterns to solve their problems.

Mathematical Structures in Pattern Organizations

Mathematical structures in well-organized patterns such as, Alexander's pattern Language, software pattern languages, pattern systems and pattern catalogues are introduced.

In order to facilitate the understanding of mathematical structures first we provide the definitions of *Graph theory* and generalized algebra known as *Category theory* (Fokkinga, 1992; Michael, 1995; Pierce, 1991).

Definition 1: Graph

A graph G is defined as $G = (V, E)$ by a set of nodes and a set of edges E . E is a relation over V . A labeled graph has the form $G = (V, E, f)$, where f is a function from E into a set of possible labels.

Definition 2 : Category

Formally, category \mathbf{C} can be defined as a collection of objects \mathbf{O} in \mathbf{C} , which satisfies the following axioms.

1. Unique-Type: for every pair (p, q) of objects in \mathbf{C} , there is a morphism $\mathbf{f}(p, q)$, denoted by $\mathbf{f} : p \rightarrow q$, from p to q such that $\mathbf{f} : p \rightarrow q$ and $\mathbf{f} : p' \rightarrow q' \Rightarrow p = p'$ and $q = q'$
2. Composition-Type: for every triple (p, q, r) of objects in \mathbf{C} , there is a partial operation from pairs of morphisms in $\mathbf{f}(p, q) \times \mathbf{g}(q, r)$ to morphism in $\mathbf{f} \circ \mathbf{g}(p, r)$, such that $\mathbf{f} : p \rightarrow q$ and $\mathbf{g} : q \rightarrow r \Rightarrow (\mathbf{g} \circ \mathbf{f}) : p \rightarrow r$.
3. Identity-Type: for every object p in \mathbf{C} , there is a morphism \mathbf{id}_p such that $\mathbf{id}_p : p \rightarrow p$.
4. Associativity: if $\mathbf{f} : p \rightarrow q$, $\mathbf{g} : q \rightarrow r$ and $\mathbf{h} : r \rightarrow s$, then $\mathbf{h} \circ (\mathbf{g} \circ \mathbf{f}) = (\mathbf{h} \circ \mathbf{g}) \circ \mathbf{f}$
5. Identity: if $\mathbf{f} : p \rightarrow q$, then $(\mathbf{id}_q \circ \mathbf{f}) = \mathbf{f}$ and $(\mathbf{f} \circ \mathbf{id}_p) = \mathbf{f}$.

Definition 3: Pre-category

If the requirement unique type is dropped in the definition of a category, then we get the definition of pre-category. Often we shall encounter data that forms a pre-category. However, these data also determined a category.

Definition 3: Subcategory

A subcategory of a category C is completely determined by its objects and morphisms, and C . A subcategory of a category C is a category in which each object, morphism, and identity is an object, morphism, and identity in C , and in which the typing and composition of C restricted to the objects and morphisms of the subcategory.

Definition 4: Full Subcategory

A subcategory of a category C is completely determined by its objects and C . A subcategory of C is a full subcategory of C if for each A, B in the subcategory, *all* the morphisms with type $A \rightarrow B$ in C are morphisms in the subcategory.

Related patterns form graphs whose nodes are patterns and edges are relationships. In order to prove that every graph structure is also a category we define the following objects and morphisms on graphs.

Object: is a node of the graph

Morphism: is a path of the graph, ie $f: A \rightarrow B$ means a path from A to B .

Identity: is an empty path, and

Composition: is concatenation of paths.

The following section describes how these objects and morphisms satisfy the five axioms: Unique-type, Composition-Type, Identity-Type and Associativity.

Unique-Type

- Let $f: A \rightarrow B$ and $f: C \rightarrow D$ are two paths from A to B and C to D respectively. Since the two paths (f) are identical, starting nodes and end nodes of the paths should be equivalent. This means $A = C$ and $B = D$.

Therefore *morphisms* are well typed.

Composition-Type

- Let $f: A \rightarrow B$ and $g: B \rightarrow C$ are two paths from A to B and B to C respectively. Since $(g \circ f)$ is defined as the concatenation of paths, $(g \circ f)$ is the path from A to C via B .

This means $\mathbf{f} : A \rightarrow B$ and $\mathbf{g} : B \rightarrow C \Rightarrow (\mathbf{g} \circ \mathbf{f}) : A \rightarrow C$

Identity-Type

- Since identity is an empty path, every object has an identity so that $\mathbf{id}_A : A \rightarrow A$

Associativity

- Let $\mathbf{f} : A \rightarrow B$, $\mathbf{g} : B \rightarrow C$ and $\mathbf{h} : C \rightarrow D$ are paths from A to B , B to C and C to D respectively. Then

$$\mathbf{h} \circ (\mathbf{g} \circ \mathbf{f}) = (A \rightarrow B) \circ \{ (B \rightarrow C) \circ (C \rightarrow D) \}$$

According to the definition of the *Composition-Type*, $\mathbf{h} \circ (\mathbf{g} \circ \mathbf{f}) = (A \rightarrow B) \circ (B \rightarrow D)$

$$\Rightarrow \mathbf{h} \circ (\mathbf{g} \circ \mathbf{f}) = (A \rightarrow D) \text{ -----(1)}$$

Similarly,

$$(\mathbf{h} \circ \mathbf{g}) \circ \mathbf{f} = \{ (A \rightarrow B) \circ (B \rightarrow C) \} \circ (C \rightarrow D)$$

$$\Rightarrow \mathbf{h} \circ (\mathbf{g} \circ \mathbf{f}) = (A \rightarrow C) \circ (C \rightarrow D)$$

$$\Rightarrow \mathbf{h} \circ (\mathbf{g} \circ \mathbf{f}) = (A \rightarrow D) \text{ -----(2)}$$

$$(1) \text{ and } (2) \Rightarrow \mathbf{h} \circ (\mathbf{g} \circ \mathbf{f}) = \mathbf{h} \circ (\mathbf{g} \circ \mathbf{f})$$

This means morphisms are associative.

- Let $\mathbf{f} : A \rightarrow B$, then

$$(\mathbf{id}_A \circ \mathbf{f}) = (A \rightarrow A) \circ (A \rightarrow B) = A \rightarrow B \text{ -----(1)}$$

$$(\mathbf{f} \circ \mathbf{id}_B) = (A \rightarrow B) \circ (B \rightarrow B) = A \rightarrow B \text{ -----(2)}$$

$$(1) \text{ and } (2) \Rightarrow (\mathbf{id}_A \circ \mathbf{f}) = \mathbf{f} = (\mathbf{f} \circ \mathbf{id}_B)$$

Since the objects and morphisms defined on graphs satisfy all axioms of a category, in general, every graph with the above objects and morphisms forms a Category.

This section describes how patterns in existing organizations form above mathematical structures: graphs and categories.

Pattern Languages

Typically pattern languages are organized from large-scale patterns to small-scale patterns based on the relationships between the patterns. Alexander's pattern language (Alexander, 1979) is organized based on the *Uses* and *Alternatives* relationships. His language starts with *Independent Regions* pattern and it addresses the construction problem as a whole and provides a partial solution by resolving some forces. Then it directs users to next pattern or patterns, sub-patterns, to use to resolve the remaining forces. These patterns will in turn provide solutions exposing sub-problems and sub-patterns to solve them. Each pattern in this language relates one

another by *Uses* and *Alternatives* relationships. Software Pattern Languages are also organized based on the *Uses* relationship but with less number of patterns. They also provide guidelines to users indicating which pattern or patterns to apply next. So structure of a pattern language is a directed or labeled graph with few cycles. The patterns and relationships between them represent the nodes and edges of the graph. A part of *A Pattern Language* is shown in Figure 2. This graph depicts patterns that can be used for building and town construction. In applying these patterns, designers can start with a pattern in top of the diagram and then apply the patterns directed from that pattern and related with the uses relationship. For instance, in order to design subculture boundary first apply the Subculture Boundary pattern and then Still Water, Access to water, Parallel roads, Industrial ribbon and Work community. It also provides alternatives such as Ring roads, Accessible greens and so on.

Software pattern languages have been documented following the Alexander's languages. However, almost all of the software pattern languages are documented with a fewer number of patterns and they address only a particular section of the application domain. Software pattern languages available in the literature provide solution to subproblems of software design. For example such languages provide solutions for web designing, GUI designing and so forth. Even though they are small compared to the Alexander's pattern language they also form similar graph structures. So pattern languages form categorical structures.

Pattern Catalogues and Systems

Since patterns do not exist isolation, they relate one another in different ways. Members of the pattern Catalogues and pattern systems can be combined based on such relationships. So, individual patterns and systems also form a number of labeled graphs but each of them with a number of different labels. For example, Figure 3 and Figure 4 show how individual patterns and pattern systems form such graphs. Figure 3 depicts a collection of patterns that provide alternative solutions to the same problem. Patterns in the Left hand side and the right hand side provide different solutions for the problem addressed by the Proxy pattern. Any of these patterns can be used to solve the problem in different contexts. Figure 4 shows a collection of patterns that relate one another in various ways. Patterns in these Figures form networks of patterns and relationships. So they also form categorical structures.

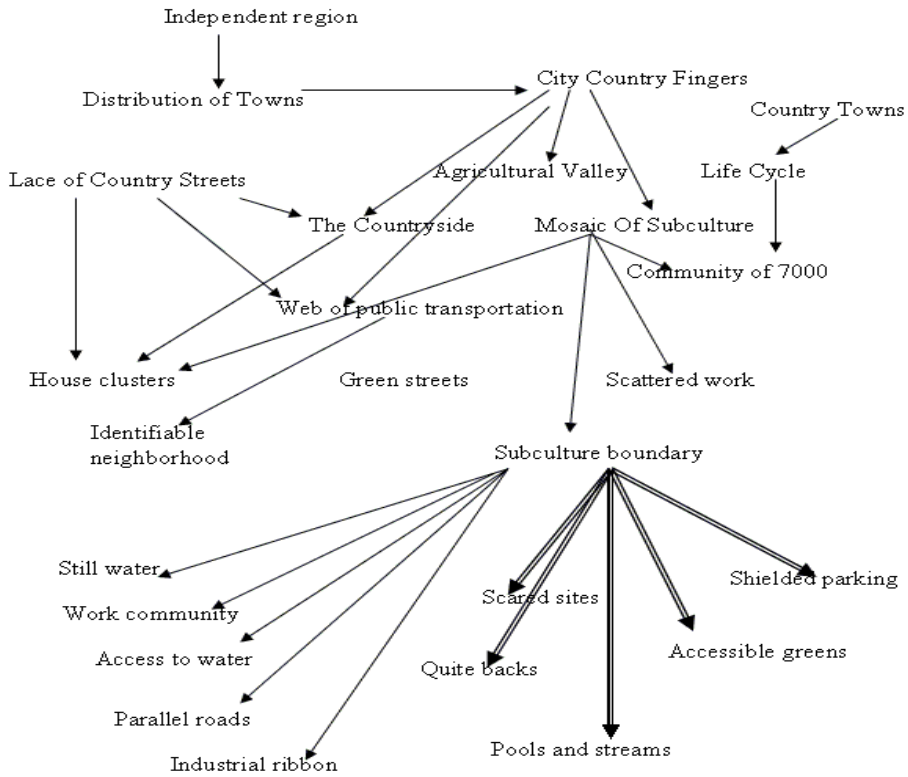


Figure 1: A portion of the structure of the *A Pattern Language*

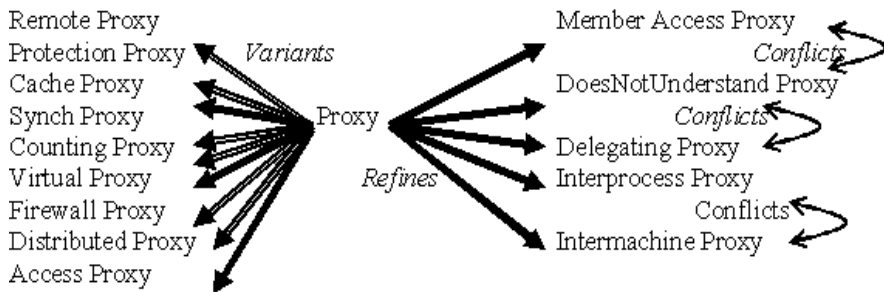


Figure 2: Illustration of individual patterns

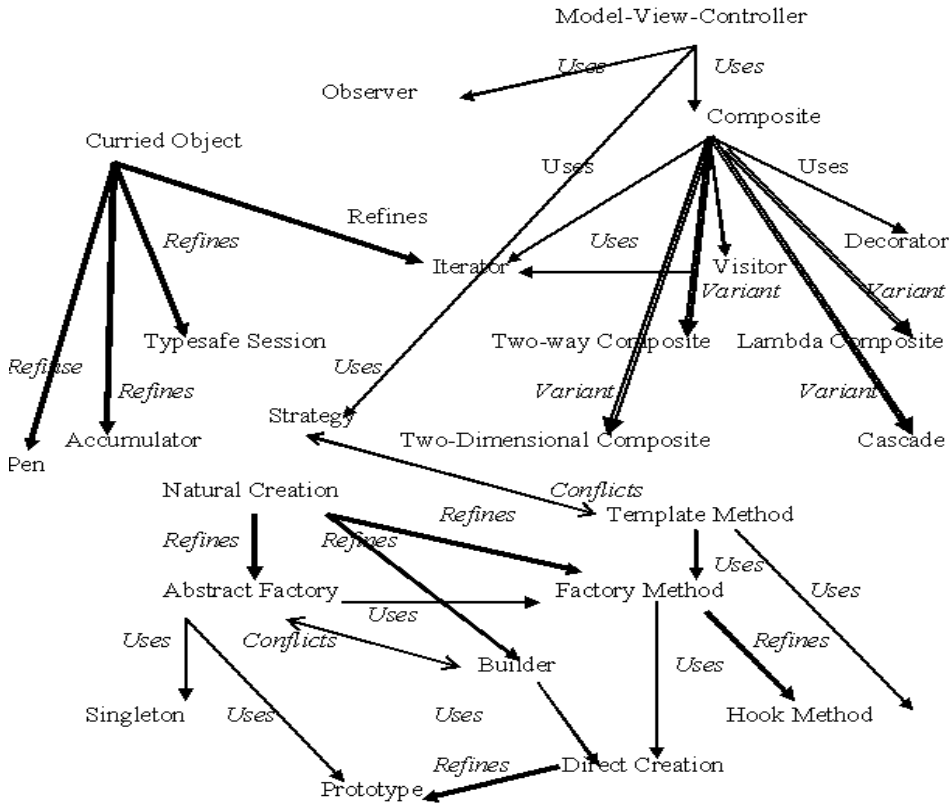


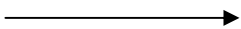

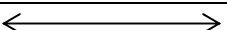

Figure 3: Illustration of individual patterns and pattern system

According to the definition of a category patterns organized into categories, systems and languages form categorical structures. These structures are not visible, because the relationships among patterns in such organization are not clearly discussed or identified.

Organizing Patterns into Sequences

This section presents the proposed organization of patterns. The structure of the organization consists of large-scale categories and small-scale categories. Small categories are derived from the large-scale categories. Since the purpose of this organization is to regroup patterns scattered through the literatures as individual patterns and pattern systems, a collection patterns systems available in (Buschnann, 1996) and a collection of individual patterns in (Riehle *et al.*, 1998; Gamma *et al.*, 1995) are used to illustrate the proposed organization In organizing patterns into

sequences, first relationships among patterns are identified. Then large scale categories are constructed based on the identified relationships. Finally large scale categories are decomposed into more meaningful sub categories. In order to distinguish different relationships among patterns the graph structures are depicted with following notations.

Relationship	Notation
Uses	
Specialization (Refines)	
Alternatives (Conflicts)	
Variants	

2.4.1 Large Scale Categories

In general, the *Uses* relationship is used to organize pattern languages. Patterns in Catalogues and systems are related by a number of relationships. As a result, patterns which address a particular problem and its sub-problems are related to one another by a number of relationships. The patterns can be organized into graphs and, hence into categories of patterns as follows.

Objects: are patterns, which can be connected by **relationships between them** to form a graph

Morphism: a pair of patterns (A, B) with A *Relates* B (a path from A to B)

Identity: $\text{id}_A = (A, A)$ (empty path)

Composition: $(A, B) \circ (B, C) = (A, C)$

Large-scale structures constructed from pattern languages, pattern catalogues and pattern systems are discussed in the proceeding sections.

Small-scale categories

The large-scale category contains patterns, which are related by a number of relationships. This category can be further organized into a set of sub-categories. Once

the patterns have been organized into categories, they can be further categorized according to the following structures.

Major Categories - Categories based on Uses/Is used relationship

Some of the patterns in the large scale categories are related by *Uses* relationship. These patterns can be organized into a separate category, which provides a set of patterns, which can be used to solve a particular problem. If this category of patterns completely solves the problem, it can be extended to a pattern language by adding rules and guidelines. Otherwise it becomes an incomplete language but it can be extend to a pattern language by adding patterns are being written. Define the category as follows:

Objects: are objects in the category defined in section 4.3 which are connected by the **Uses / Is used** relationship

Morphism: a pair of patterns (A, B) with A Uses/Is used B

Identity: $\text{id}_A = (A, A)$

Composition: $(A, B) \circ (B, C) = (A, C)$

Alternative Categories

Patterns that are not related with the *Uses* relationship are also available in large scale categories. These patterns are also derived and organized into subcategories. Each such category provides a collection of alternative patterns to the members of the major categories. So they assist users in selecting the most appropriate pattern among a collection. Three types of alternative categories are derived based on the *Specializes*, *Variants* and *Conflicts* relationships.

Categories based on Specializes/Generalizes relationship

The patterns related by *Specialization* relationship can be organized into another category or a set of categories. Each of these categories provides specialization of the patterns in pattern languages or major categories. This category helps user to find the general and specific solution of a pattern. The category can be formed as follows:

Objects: are objects in the category defined in section 4.3 which are connected by the **Specialization / Generalization** relationship

Morphism: a pair of patterns (A, B) with A Specializes (Refines) B

Identity: $\text{id}_A = (A, A)$

Composition: $(A, B) \circ (B, C) = (A, C)$

Categories based on Alternatives (Conflicts) relationship

A set of patterns related by *Alternatives* relationship provides alternative solutions to similar problems. Such patterns can be organized into a separate category and such an organization provides the alternative patterns to a particular pattern in pattern languages and major categories. This assists users to select the most suitable pattern(s) from a set of patterns, which are applicable to a particular problem. The category is defined as follows:

Objects: are objects in the above category (5.1) which are connected by the **Alternatives (Conflicts)** relationship

Morphism: a pair of patterns (A, B) with A Conflicts B

Identity: $\text{id}_A = (A, A)$

Composition: $(A, B) \circ (B, C) = (A, C)$

Categories based on Variants relationship

The categorization of patterns based on the *Variants relationship* helps user to find possible variants of a particular pattern. So, the patterns in this category provide alternative solutions to the same problem or a common solution to a number of different problems. Therefore, user can select an appropriate variant of a pattern to solve the problem at hand. The category is formed as follows:

Objects: are objects in the above category (5.1) which are connected by the **Variants** relationship

Morphism: a pair of patterns (A, B) with A is a variant of B

Identity: $\text{id}_A = (A, A)$

Composition: $(A, B) \circ (B, C) = (A, C)$

These categories satisfy the conditions mentioned in the definition 3. As such, these are subcategories of the category defined above.

Application of the methodology

We have applied this technique to organize patterns in **GOF** catalogue (Gamma *et al.*, 1994), **Patterns of Software Architecture** (Buschmann, 1996) and some individual patterns derived from the literature to illustrate the method. These patterns form a category shown in Figures 3 and 4. These categories are partitioned into subcategories according to the type of relationships. More precisely a collection of patterns related with a particular type and formed a graph structure are derived and organized into a separate subcategory. In this way several major categories and alternative categories are derived. Finally each major category along with related alternative categories is organized into a pattern sequence. In order to save the space one of the resultant major categories and some of the alternative categories are shown. Other categories are listed with members only.

Deriving Major Categories

In the categories shown in Figures 3 and 4, there are two sets of patterns that are related with the Uses relationship can be identified. One set consists of Template Method, Hook Method, Abstract Factory, Prototype, Factory Method, Singleton, Builder and Direct Creation. This set is organized into a major category and is shown in Figure 5. The other one consists of Model-View-Controller, Observer, Composite, Strategy, Decorator, Visitor and Iterator. It is organized into another major category.

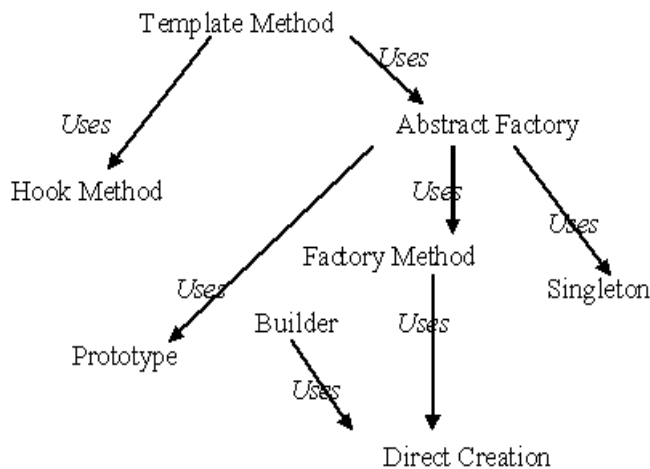


Figure 4: Example-structure of a category based on *Uses* relationship

Deriving Alternative Categories

In the categories shown in Figures 3 and 4, there are four sets of patterns that are related with the Refines relationship can be identified. These are organized into separate alternative categories. First category consists of Proxy, Member Access Proxy, DoesNotUndertandProxy, Distributed proxy, Delegating Proxy, Interprocess Proxy and Intermachine proxy. This category is shown in Figure 6. Other categories consist of patterns listed below.

{Curried object, Pen, Iterator, Typesafe session and Accumulator}

{Natural creation, Abstract Factory, Factory method, Builder, Hook method}

{Direct creation, Prototype}

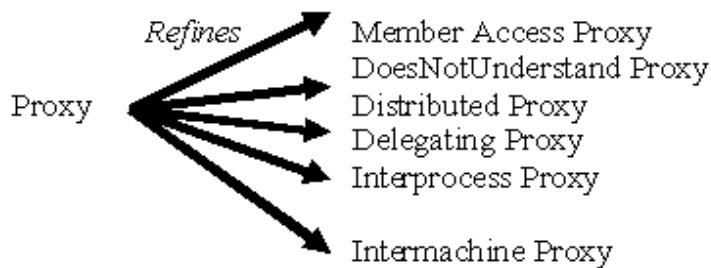


Figure 5: Example-structure of a category based on *Refines* relationship

In the category shown in Figures 3 and 4, there are three sets of patterns that are related with the Conflicts relationship can be identified. These are organized into separate alternative categories. First category consists of Member Access Proxy, DoesNotUndertandProxy and Delegating Proxy. This category is shown in Figure 7. Other categories consist of patterns listed below.

{Strategy and Template Method}

{Abstract Factory, Builder}

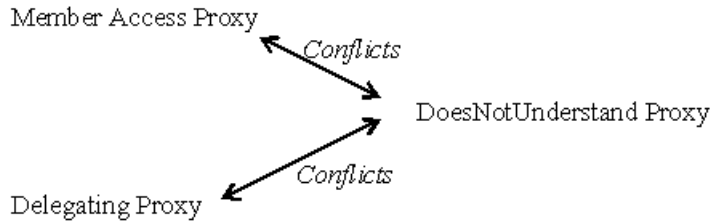


Figure 6: Example-structure of a category based on *Conflicts* relationship

In the category shown in Figures 3 and 4, there are two sets of patterns that are related with the Variant relationship can be identified. These are organized into separate alternative categories. First category consists of Proxy, Access Proxy, Remote Proxy, Protection Proxy, Cache Proxy, Synch Proxy, Counting Proxy, virtual Proxy, Firewall Proxy, Lazy Proxy and Distributed Proxy. This category is shown in Figure 8. Other categories consist of patterns listed below.

{Composite, Two-Way Composite, Lambda Composite, Two-Dimensional Composite and Cascade}

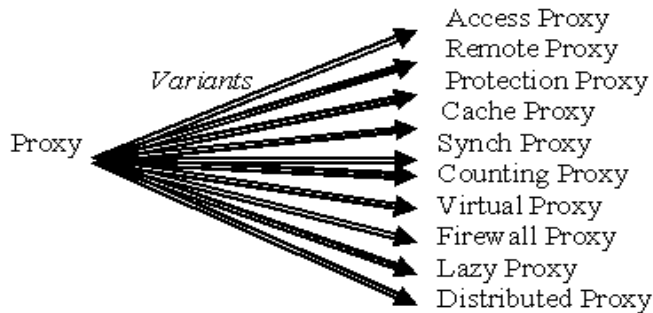


Figure 7: Example-structures of the categories based on *Variants* relationship

In applying the methodology to categories shown in Figures 3 and 4, two major categories and nine alternative categories are obtained. The next step is to organize them into sequences. Since there are two major categories, two pattern sequences can be formed. They can be formed by combining each major category with the related alternative categories.

One major category consists of Template Method, Hook Method, Abstract Factory, Prototype, Factory Method, Singleton and Direct Creation. Among the

resultant alternative categories select the categories that have members of this major category. Alternative categories {Natural creation, Abstract Factory, Factory method, Builder, Hook method} and {Direct creation, Prototype} contain some members of this major category. Therefore the first sequence consists of these three categories. That is:

Sequence 1

Major category – {Template Method, Hook Method, Abstract Factory, Prototype, Factory Method, Singleton and Direct Creation}

Subcategories - {Natural creation, Abstract Factory, Factory method, Builder, Hook method} and {Direct creation, Prototype}

The second major category consists of Model-View-Controller, Observer, Composite, Strategy, Decorator, Visitor and Iterator. Alternative categories {Strategy, Template Method} and {Composite, Two-Way Composite, Lambda Composite, Two-Dimensional Composite, Cascade} contain members of this major category. Therefore the second sequence consists of these three categories. That is:

Sequence 2

Major category – {Model-View-Controller, Observer, Composite, Strategy, Decorator, Visitor and Iterator}

Subcategories - {Strategy, Template Method} and {Composite, Two-Way Composite, Lambda Composite, Two-Dimensional Composite, Cascade}.

DISCUSSION

Since pattern languages address the pattern selection problem by providing rules and guidelines, pattern languages are the best organization method among existing organization methods. Uses or Combines relationship connects each pattern in a pattern language to one another. Therefore, pattern language forms a directed-labeled graph. We have proved that every graph is a category.

In contrast, individual patterns exist in isolation but they are implicitly related to one another in different ways. These relationships are used to connect the patterns such a way that they also form categories. These categories are the large-scale structures of the pattern organization. These structures may contain a number of sequences of patterns which are related one another by *Uses* relationships. Since, patterns connected by the *Uses* relationship can be extended into pattern languages, they are organized into subcategories and such categories are the major small-scale

structures of the organization. Each pattern in these major categories is linked to another category, which contains variants of the pattern. These secondary categories help to find alternative patterns.

I have applied this technique to organize patterns in the literature. The application results in two major categories and nine alternative categories. This results in two sequences. Practitioners can first search the sequences to find a collection of related patterns to solve their problems. Users can select entire major category of patterns or a subset of a major category, which can be collectively applied to solve a particular design problem. If users need to find alternative patterns, they can be found from the related alternative categories available in the sequence. This organization reduces the amount of searching for applicable patterns. It also addresses the pattern selection problem up to some extent. However, this organization technique needs to be optimized with indexing or reasoning technique because users need to select an appropriate sequence for a particular application domain or a problem.

Conclusions and Future Work

In this paper, I have discussed the mathematical structures in existing pattern organization techniques, and the common relationships between patterns. I have also described the mathematical structures in current organization methods, and have introduced a new organization technique based on the mathematical structures in efficient pattern organizations. This technique organizes the related pattern into pattern sequences. Each sequence consists of a category based the *Uses* relationship and a collection of related alternative categories. New organization help both experienced and novice users to find a set of related patterns among the existing patterns. In effect, users can easily find a collection of related patterns to a particular problem. The benefits of this organization method can be summarized as follows.

1. Practitioners can find a collection of related patterns from the sequences that provide patterns related with the *Uses* relationship and possible alternatives to those patterns.
2. Major and alternative categories facilitates the selection of the most appropriate collection of patterns to a problem quickly and easily.
3. The applicability of the proposed organization could be optimized by incorporating an indexing or reasoning technique.

REFERENCE

- Alexander, C., M. Silverstein & S. Ishikawa. 1977. A Pattern Language. Oxford University Press, first edition.
- Alexander, C. 1979. The Timeless Way of Building, Oxford University Press, first edition.
- [Barr](#), M. & [C. Wells](#). 1995, Category Theory for Computing Science, Prentice Hall; 2nd edition.
- Coplien, O. 1996. Software Patterns a White Paper. SIGS publications.
- Coplien, O. & D. Schmidt. 1995. Pattern Languages of Program Design, Addition-Wesley, first edition.
- Fokkinga, M.M. 1992. A Gentle Introduction to Category Theory, University of Utrecht.
- Fowler, M. 1997. Analysis Patterns: reusable object modules. Addition-Wesley, first edition.
- Gamma, E., H. Richard, R. Johnson & J. Vlissides. 1994. *Design patterns elements of reusable object oriented software*. Addition-Wesley, second edition.
- John, M., J. Coplien, J. Vlissides & L.K. Norman, 1996. Pattern Languages of Program Design 2. Addition-Wesley, first edition.
- Lavender, R. G. & D. Schmidt. 1996. Active object: A behavioral pattern for concurrent programming. Pattern Languages of Program Design 2, 483-500.
- Levine, D. L., C. D. Gill & D. Schmidh. 1999. Object Lifetime Manager: A complementary pattern for controlling object creation and destruction. Submitted to 5th Pattern Languages of Programming Conference.
- Noble, J. 1998. Classifying relationships between object-oriented design patterns. In Australian Software Engineering Conference (ASWEC).
- Pierce, B. C. 1991, [Basic Category Theory for Computer Scientists](#), The MIT Press.
- Pree, W. 1994. Design Patterns for Object Oriented Software development. Addition-Wesley, first edition.
- Richard, G. Patterns Definitions. <http://hillside.net/patterns/definitions.html>.
- Riehle, D., R. Martin & F. Buschmann. 1998. Pattern languages of program design 3. Addition-Wesley, first edition.
- Rising, L. 2000. The Pattern Almanac 2000, Addition-Wesley, first edition.

- Rohnert, H., F. Buschmann, M. Regine & P. Sommerland. 1996. A System of Patterns. Addison-Wesley, first edition.
- Schmidt, D. & C.D. Cranor. 1996. Half-sync/Half-async: An architectural pattern for efficient and well-structured concurrent I/O. Pattern Languages of Program Design 2 : 437-460.
- Zhao, L. & T. Foster. 1998. Plots: A Pattern Language of Transport Systems- Point and Route. Pattern Language of Program design 3.
- Zimmer, W. 1995. Relationships Between Design Patterns. Pattern Languages of Program Design 1: 345-364.