

Semantics of a Pattern System

Ashraf Gaffar and Naouel Moha

GEODES - Group of Open and Distributed
Systems, Experimental Software Engineering

Concordia University, Quebec, Canada
University of Montreal, Quebec, Canada

E-mail: gaffar@cs.concordia.ca, mohanaou@iro.umontreal.ca

Abstract

The wide acceptance of the “Design Patterns” [5] has encouraged experts in other software domains to formulate their experience into pattern format hoping to make it readily reused by developers. We now have numerous pattern collections covering all aspects of software development from analysis to deployment and refactoring. But developers can be overwhelmed by this large number and the lack of coordination and in consistencies among them. These patterns have many similarities and redundancies which may contribute to misunderstanding and wrong reuse. Some research has proposed standards to writing patterns [1] [7] but they were rarely used because each pattern author prefers to use their own creativity [3] which is often a good thing. We propose another approach to address this problem. In each specific software domain, we collect and pre-process existing patterns by defining, detecting and removing some kinds of redundancies between them. The result is a smaller collection of patterns from different sources that have fewer redundancies which reduces confusion and promotes the proper reuse.

Keywords: Design Patterns, Models, Redundancies, Pattern Relationships, Semantics, Similarities.

1. Semantics of a Pattern System

We show some semantics associated with the concept of Generic Pattern Model and some semantics of equivalent/identical relationships between patterns. We originally applied them to interaction design patterns but they can be extended to other types of patterns. They are showing possible benefits in form of tool-support on top and independent of the core pattern information.

1.1. The eXtensible Minimal Triangle, XMT

Data models generally have syntax and semantics. The XML handbook [6] explains that semantics can be further divided into *semantic labeling* of contents, and *abstract interoperable behavior* as demonstrated in figure 1.

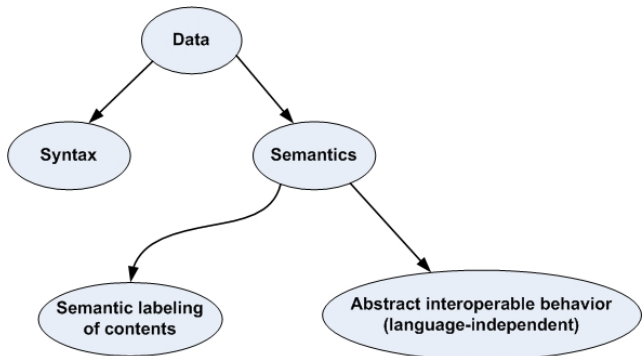


Figure 1. The Data Constituents

In the GPT (Generic Pattern Type) model [2], we defined the syntax of a generic pattern model, and the semantic labeling used for its syntax. Our work here focuses on the abstract interoperable behavior, which involves the behavior underlying the meaning of some of the tags we are using. We start by some preconditions and definitions.

As frequently mentioned in the pattern community, the common denominator of all pattern definitions is “a problem to a solution in a context”. Based on that, we defined the minimal triangle as in figure 2.

Definition. *The Minimal Triangle* is a representation of a pattern that has the three elements: a problem, a context, and a solution. No other elements are

present in the minimal triangle. The minimal triangle represents the core meaning of a pattern. Any missing element of the three will result in a trivial pattern.

Definition. A *trivial pattern* is a pattern that has at least one empty element from its minimal triangle.

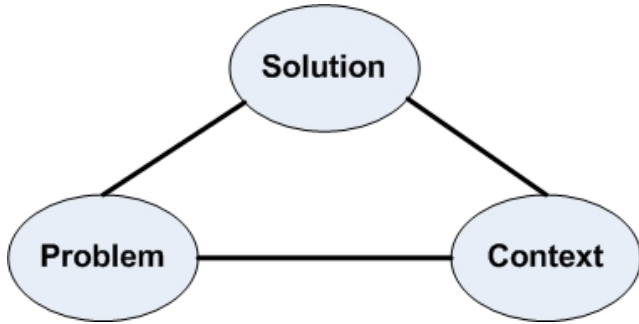


Figure 2. The Minimal Triangle

1.2. Identical Patterns

Informally, identical patterns are the same patterns mentioned in different collections. To be able to use some of our models, we introduce the following formal definitions:

- **Precondition:** Each pattern has a unique ID.
- **Precondition:** Each non-trivial pattern has an existing, non empty minimal triangle (this can be inferred from the definition of trivial pattern).
- **Precondition:** The following definitions strictly apply to non-trivial patterns. For trivial patterns, the definitions can be ambiguous.
- **Precondition:** $Pattern_name(A)$ refers to a single object called “the name of pattern A”, of type “string”.
- **Precondition:** $Pattern_ID(A)$ refers to a single object called “the identification of pattern A”, of type “string”.
- **Precondition:** $Pattern_aliases(A)$ refers to the set of zero or more objects called “the aliases of pattern A”, of type “set of string objects”.
- **Precondition:** There is a one-to-one correspondence between $Pattern_name(A)$ and $Pattern_ID(A)$. We emphasize the word “correspondence” to denote a “one-to-one” and an “onto” function between $Pattern_name(A)$ and

$Pattern_ID(A)$. In other words, the inverse relationship between $Pattern_name(A)$ and $Pattern_ID(A)$ is also a one-to-one function.

- **Notation (from the set theory):** $(A) \in (B)$ refers to the object A being an element of the set of objects B.

These are important preconditions that we will use in our next definitions to disambiguate some patterns that are identical or similar at different degrees. Some patterns are already connected to others, but the majority of them are not. We have identified many of these redundancies.

Definition. *Identical Pattern*

Pattern A is identical to pattern B, written $A = B$, if¹

$Pattern_name(A) = Pattern_name(B)$	or
$Pattern_name(A) \in Aliases(B)$	or
$Pattern_name(B) \in Aliases(A)$	

1.3. Similar Patterns

As in several other applications, it is sometimes useful to separate between the look (the presentation) and the behavior. The same concept is used in separation of java’s look and feel, contents and presentation in CMS (Content Management Systems) and other models. We define the next pattern relationships based on a similar concept: Patterns that look similar and act similar (identical- and similar patterns), and patterns that look different but act similar (equivalent patterns).

- **Precondition:** The next definition is based on the definition of the *Minimal Triangle*.
- **Definition:** The XMT repository specifies three finite sets of keywords corresponding to the three MT items:
 - The first set defines the specific kinds of **problems** covered by the patterns in form of reserved keywords.
 - The second set defines the specific kinds of **contexts** covered by the patterns in form of reserved keywords.
 - The third set defines the kinds of **solutions** covered by the patterns in form of reserved keywords.

¹This is a sufficient, but not a necessary condition. We can make it necessary by augmenting the aliases set of each pattern with our findings of identical patterns.

- **Definition:** The *Extensible Minimal Triangle* model refers to the three parts *problem, context, and solution* of a given pattern as presented in the GPT and using a subset of reserved keywords elaborated in the *repository* of the XMT model.

The extensibility comes as a key concept of the XMT. If new patterns are to be added, and the keywords of each of its MT parts are not in the space of the XMT, the XMT must be extended by carefully defining the new keywords and adding them to the XMT repository. Using reserved keywords for context, problem, and solution allows for automated text processing, a scalable solution. However, the three items of the MT will have to be divided into a “brief” part, containing the reserved keywords, and an elaborate part containing the explanation and details of each item. This makes it suitable for human reading as well as machines, and is sometimes applied to the solution item by providing a thumbnail solution “solution-brief” and an elaborate solution.

- **Definition:** *Similarity Criteria* is a set of logical conditions that decide if a pattern A is similar to a pattern B.
- **Definition:** *Similar Patterns* (a similarity criterion)

Pattern A is similar to pattern B, denoted $A \equiv B$, if²

$$MT(A) = MT(B)$$

(where MT denotes the minimal triangle)

- **Definition:** *Equivalent Patterns* (a similarity criterion)

Pattern A is equivalent to pattern B, denoted $A \approx B$, if

$$\begin{aligned} \text{Problem}(A) &= \text{Problem}(B) \text{ and} \\ \text{Context}(A) &= \text{Context}(B) \end{aligned}$$

We can see that an equivalence relationship is a superset of a similar relationship.

Two issues arise:

- Identical patterns are to be determined by pattern author (by including aliases to a pattern either to refer to another known name, or another pattern), but will also be complemented by our Structured Expert Support (SES) as shown in [2].
- Similar and equivalent patterns can be determined within the activity of SES in two different ways:

- **Formally**, by comparing the three MT items and applying the definitions of similarity given above. Once patterns are modulated according to the XMT model, this process can be automated using simple tools.
- **Informally**, by manually selecting patterns according to SES similarity and redundancy criteria.

2. Structured Expert Support Implementation

Beside models, manual work is essential to both clearing out, and uploading modeled patterns. Here we create new assumption set, and define some activities to help build the new system. To address the relationships between patterns as explained in the extrinsic data part within the GPT [3], we have defined two types of relationships:

- **Structural relationships:** These are patterns that have some common structure (like a common MT; or part of it).
- **Assimilation relationships:** Those patterns are considered from the design process point of view. Two patterns that are completely different in structure can still have an assimilation relationship, like complementing or competing with each other.

2.1. Argument

These two types might look orthogonal to each other, but -generally speaking- we assert that structural relationships should be included as a subset of all legitimate assimilation relationships. During the design process, patterns can be replaced based on several criteria. Similarity of pattern structure is a criterion that warrants the possibility of replacement. This concept will be demonstrated with some examples later in the paper. We will discuss this further with the issue of redundancies and show the unwanted effect of structurally entwined patterns.

Gamma et al. [5] emphasize in their book “Design Patterns” that defining the contextualized relationship between patterns is a key notion in the understanding of patterns and their usages. Zimmer [8] implements this idea by dividing the relations between the patterns of the Gamma catalog itself in 3 types: “X is similar to y”, “X uses Y”, and “Variants of X uses Y”. Based on Zimmer’s work, we showed in [4] some additional relationships between patterns from an assimilation point

²As in 1, this is a sufficient, but not a necessary condition

of view, not a structural one.
 We restate them here:

- **Subordinate (X, Y)** if and only if X is embeddable in Y. Y is also called superordinate of X.
- **Equivalent (X, Y)** if and only if X and Y can be replaced by each other.
- **Competitor (X, Y)** if X and Y cannot be used at the same time for designing the same artifact.
- **Neighboring (X, Y)** if X and Y belong to the same pattern category (family) or to the same design step as the described pattern.

A major nuisance to our work on pattern relationships, and certainly that of other users is the “noisy similarities” between patterns. We can visualize the relationships between patterns as in figure 3. Part of the similarity is a healthy relationship of **similar** or **equivalent** patterns that can easily replace each other; represented by the intersection area of the two ellipses. Several dissimilar patterns have relationships like **competitor**. A considerable part of similarities, however, has redundancies in it, and we excluded them from the concept of pattern relationships. They are represented in the part of the “pattern similarities” ellipse outside the relationships ellipse. According to our investigation on many patterns, it is not easy to decide on the nature of the relationships in these cases.

To demonstrate, if we said that pattern A is a competitor to pattern B, and then we have another pattern C which is slightly (or -in general- vaguely) similar to pattern B, can we say that pattern C is also a competitor to pattern A. The answer is that we have to resolve the vague similarity between B and C to remove this ambiguity first. Logically speaking, this will not guarantee a solution to the question. But if we were able to assert -for example- that pattern B and pattern C have an inheritance relationship, we might assume that A and C are likely competitors as well. If we were able to assert that pattern B and pattern C are similar, then we can remove pattern C altogether. In many cases we found that some patterns contain a full detail of other patterns with a slight addition and often with a slight omission. Other similarities have more entwined structure that has no clear answer. We provide more discussion in the next section.

2.2. Inter-collection Redundancies ICR

Besides our automatic discovery approach through the XMT model, similar patterns have been discovered

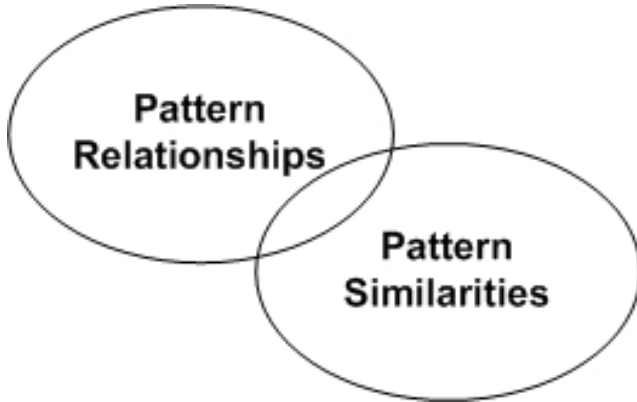


Figure 3. Useful and noisy similarities

and recorded manually. These two approaches are going in parallel to complement each other as explained earlier. We have identified many identical patterns and similar patterns with varying degrees of similarity. In some cases we were able to identify patterns in different collections that are exactly the same, but were presented by different authors under different names, despite their identical analysis. This is an “easy work”, and these are marked simply for removal. Other identical patterns are more difficult to locate.

The challenge comes with what we define as *entwined patterns*. They represent another case of partially similar patterns that can be confusing and has to be resolved. It is when two (or more) patterns have common features but each one still has a unique set of features. Despite the several types of entwining - depending on the type of redundant features- they can all be logically represented by the case of entwined brackets:

$$\{PartOfPatternA(CommonPart)PartOfPatternB\}$$

In several domains, like in programming languages, this is generally not allowed. In pattern domain, our observations show that this is one of the most confusing redundancies between patterns. We need to dissolve these patterns either by combining them into one pattern or separating them into two distinct patterns.

Note: Analogous to the defined relationships of inheritance (is-a) and aggregation (has-a) in object oriented programming, we also have pattern inheritance and aggregation relationships. In patterns domain, these are not a problem of similarity or a source of confusion; the former is addressed in a hierarchical model. The latter is a clear case of assimilation relationship. Connecting these patterns together is done through the extrinsic data part of the GPT.

3. Conclusion

In this paper we addressed the problem of redundancies between patterns and its negative effect on correct pattern reuse. It is hard and impractical to ask pattern users to observe and avoid redundancies among them. A more feasible approach is to detect and analyze these redundancies and try to reduce them. We selected interaction design patterns as a start and collected many of them. After detailed comparison and analysis, we identified several types of redundancies ranging from identical patterns (100 percent similar) into lower degrees of similarity and we grouped them into different types. We then formally defined these types in abstract formats to help reapply them on other types of patterns. We used auxiliary models to help apply the similarity criteria in an automated way as well as manually. In the future, we are extending our work on two fronts:

1. We are growing the eXtensible Minimal Triangle to add more standardized keywords to help automate the detection process of pattern similarities/redundancies.
2. We are applying the abstract concept of redundancies in other domains of software patterns.

References

- [1] S. Fincher and J. Finlay. Chi 2003 report: Perspectives on hci patterns: Concepts and tools; introducing plml. *Interfaces, the international journal of human computer interaction*, 56:26, Autumn 2003.
- [2] Ashraf Gaffar. The 7c's: An iterative process for generating pattern components. In *proceedings of HCI International, the 11th International Conference on Human-Computer Interaction*, 2005.
- [3] Ashraf Gaffar, Ahmed Seffah, and John Van der Poll. Hci patterns semantics in xml: A pragmatic approach. In *proceedings of of HSSE '05, International workshop on Human and Social Factors of Software Engineering, in ICSE 2005, the 27th International Conference on Software Engineering*,. ACM publishing, New York, NY, US, 2005.
- [4] Ashraf Gaffar, Daniel Sinnig, Ahmed Seffah, and Peter Forbrig. Modeling patterns for task models. In *proceedings of TAMODIA, 3rd International Workshop on Task Models and DIAGrams for user interface design*, 2004.
- [5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1st edition, 1994. ISBN: 0-201-63361-2.
- [6] Charles F. Goldfarb and Paul Prescod. *Charles F. Goldfarb's XML handbook*. Charles F. Goldfarb definitive XML series. Fifth edition, 2004. ISBN: 0-13-049765-7.
- [7] G. Meszaros and J. Doble. A pattern language for pattern writing, Mai 5th 1997.
Available at: <http://hillside.net/patterns/writing/patternwritingpaper.htm>.
- [8] Walter Zimmer. Relationships between design patterns. pages 345–364, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN: 0-201-60734-4.