Coupling of Design Patterns: Common Practices and Their Benefits

William B. McNatt James M. Bieman
Computer Science Department
Colorado State University
Fort Collins, CO 80525 USA
McNatt: +1-303-971-2271, Bieman: +1-970-491-7096
william.b.mcnatt@lmco.com, bieman@cs.colostate.edu

Abstract

Object-oriented (OO) design patterns define collections of interconnected classes that serve a particular purpose. A design pattern is a structural unit in a system built out of patterns, not unlike the way a function is a structural unit in a procedural program or a class is a structural unit in an OO system designed without patterns. When designers treat patterns as structural units, they become concerned with issues such as coupling and cohesion at a new level of abstraction. We examine the notion of pattern coupling to classify how designs may include coupled patterns. We find many examples of coupled patterns; this coupling may be "tight" or "loose", and provides both benefits and costs. We qualitatively assess the goodness of pattern coupling in terms of effects on maintainability, factorability, and reusability when patterns are coupled in various ways.

Keywords: Design patterns, object-oriented design, design quality, coupling.

1. Introduction

Design patterns are architectural units, just as classes are implementation units. We can view a system as a collection of interacting patterns and independent classes. Pattern instance attributes are potentially measurable in a manner similar to class attributes. Thus, we can examine notions such as pattern coupling.

Pattern coupling results from connections between patterns. Common classes can connect two patterns. They play roles in more than one pattern by referencing common objects, and by using methods in another pattern. Figure 1 shows a UML class model of two coupled patterns from the well-known book by Gamma et al [8]. In this example, an instance of an Abstract Factory pattern is coupled with an instance of a Visitor pattern through shared classes. Interface ASTNode, classes AST, StmtNode, and DeclNode play roles in both patterns. A change in parts of one pattern can affect the behavior of the other pattern.

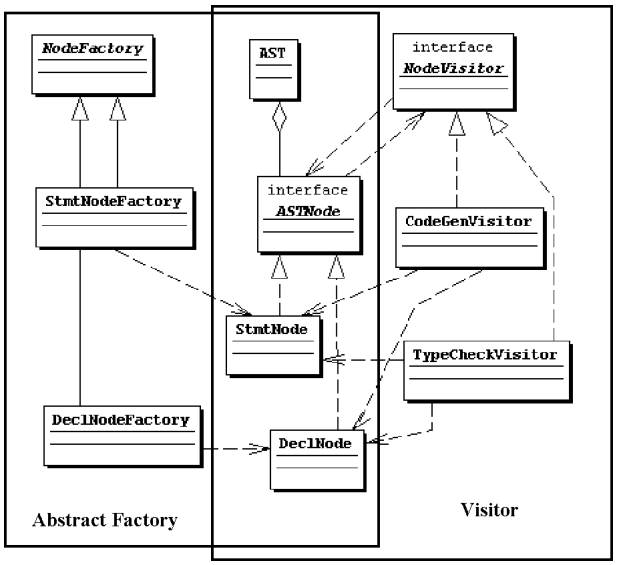

Figure 1. Overlapping patterns: Abstract Factory and Visitor.

Although we have not found the notion of pattern coupling discussed in the literature, we found numerous examples of coupled patterns. In this paper, we study the 23 patterns described in Gamma et al [8]. The design pattern literature includes many descriptions of actual implementations that make use of patterns and pattern languages. Published examples include coupled patterns that are in practical use today. Review of the pattern coupling techniques used in practice can reveal characteristics of this coupling, and help us understand when and how to couple patterns.

2. Approach

We examined the available literature describing specific design pattern applications to find examples of pattern coupling using the following process:

1. Survey recent literature concerning design pattern applications for examples of interconnected patterns.
2. Compile a list of all example instances of Gamma et al design patterns found in the literature.
3. Create a sub-list of all groups of patterns used as pairs or multiple sets of interacting Gamma et al patterns.
4. Group the list into categories of coupling types (tightly versus loosely coupled) and interaction types (intersection, composition, or embedding) with a rationale for each grouping.
5. Evaluate the identified instances of coupled patterns in terms of desirable software qualities (maintainability, factorability, reusability, and ease of implementation) with rationale for each grouping.
6. Analyze the data to better understand the pattern coupling techniques that, in general, tend to contribute to design "goodness" and which ones should be used only to satisfy specific application needs.

We included only those coupled pattern groups that consist exclusively of patterns defined in Gamma et al.

3. Study Design Patterns

The study pattern set consists of the 23 Gamma et al OO patterns. The patterns are categorized into their three main groups and listed below with descriptions quoted from Gamma et al [8]:

- Creational Patterns.

  Factory Method: "Define an interface for creating an object, but let subclasses decide which class to instantiate."

  Abstract Factory: "Provide an interface for creating families of related or dependent objects without specifying their concrete classes."

  Builder: "Separate the construction of a complex object from its representation so that the same construction process can create different representations."

  Prototype: "Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype."

  Singleton: "Ensure a class only has one instance, and provide a global point of access to it."

- Structural Patterns.

  Adapter: "Convert the interface of a class into another interface clients expect."

  Bridge: "Decouple an abstraction from its implementation so that the two can vary independently."

  Composite: "Compose objects into tree structures to represent part-whole hierarchies."

  Decorator: "Attach additional responsibilities to an object dynamically."

  Facade: "Provide a unified interface to a set of interfaces in a subsystem."

  Flyweight: "Use sharing to support large numbers of fine-grained objects efficiently."

  Proxy: "Provide a surrogate or placeholder for another object to control access to it."

- Behavioral Patterns.

  Chain of Responsibility: "Avoid coupling the sender of a request to its receiver ... Chain the receiving objects and pass the request along the chain until an object handles it."

  Command: "Encapsulate a request as an object, thereby letting you parameterize clients with different requests..."

  Interpreter: Given a language, define a representation for its grammar along with an interpreter."

  Iterator: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation."

  Mediator: "Define an object that encapsulates how a set of objects interact."

  Memento: "Capture and externalize an object's internal state so that the object can be restored to this state later."

  Observer: "Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated."

  State: Allow an object to alter its behavior when its internal state changes."

  Strategy: "Define a family of algorithms, encapsulate each one, and make them interchangeable."

  Template: "Define the skeleton of an algorithm in an operation, deferring some steps to subclasses."

  Visitor: "Represent an operation to be performed on the elements of an object structure."

These patterns represent all of the patterns that we targeted in this investigation. Although other patterns are documented in the collection of papers used in this study, we did not include them in this study. Thus, the pattern combinations discovered are limited to those that can be derived from pairs or multiples of patterns from Gamma et al [8].

4. Papers Included in the Study

Several criteria were applied to the set of papers to support consistent analyses of pattern examples:

1. Each paper must have contained at least 1 pattern from Gamma et al [8] to be included in the study.
2. Papers were not excluded from the study if they did not contain coupled pattern combinations to add to the total data set. This allowed a realistic depiction of the actual prevalence of coupling in practice.
3. Individual pattern recognition was based on the actual documentation using the name of the pattern.
4. We identify pattern combinations either from actual direct references in the research papers or from their indirect recognition after a review of the pattern structures presented in the papers.
5. The classification of the patterns into groups according to coupling type (loose vs tight) and interaction type (embedded, interactive, and composite) was accomplished via paper inspection by a single reviewer without subsequent independent validation.

The criteria, especially Criteria 1, allowed us to reject many papers as sources for data for this study. Approximately one third of the papers that we found describing pattern use was rejected due to failure to find even one instance of a documented design pattern from Gamma et al [8]. The final selected set of 16 papers used in this study consisted of a fair distribution of papers from industrial applications and analytical studies. We classified the papers into the following four categories based on the nature of the study and the source of data:

- Pure Analytical Study. Four studies consist of abstract analytical application of ideas that are not applied to real world problems [3, 4, 6, 11].
- Analytical study with synthetic examples. Three studies include synthetic examples which are abstractions of realistic problems that can occur in commercial software development [10, 13, 15].
- Industry case studies of existing code. Five studies locate existing pattern architectures, or update existing designs to make use of patterns [1, 2, 7, 9, 18].
- Industry Case Studies of new designs. Four studies examine new or recent development projects which deliberately makes use of patterns during the design phase [5, 14, 16, 17].

5. Pattern Occurrences

The occurrence of patterns in the selected papers was also quite evenly distributed among the Gamma et al patterns. Table 1 lists pattern frequencies. Some patterns appear more frequently, but all patterns are used to some extent. Certain patterns appear to be more desirable based on their frequent usage. Most of the more popular patterns support interface abstraction and reusability. One example is the Observer pattern. However, the popularity of some of the patterns may be due to the simplicity of the pattern. For example, the Singleton pattern is simple, but useful, and is often embedded in other more complex patterns.

Our data includes far more instances of Behavioral patterns than either Creational or Structural patterns. However, there are approximately twice as many Creational patterns in the Gamma et al book than the other two pattern types. The frequency totals actually correspond very well with the quantity of pattern types in each subtype. It appears that pattern usage is much more dependent on the individual pattern type than its particular subtype grouping.

6. Pattern Coupling

We identified all coupled patterns. Such coupled patterns consist of all pattern pairs or multiples that are connected. We classified the sets of connected patterns in terms of loose vs. tight coupling, and pattern interaction types:

- Coupling types.

  Loosely Coupled: Patterns are loosely tied together with few connections, making it easier to separate the patterns. They should promote quicker, simpler future design changes.

  Tightly Coupled: Patterns are tightly tied together with many links and dependencies. Minor changes to one of the patterns will be difficult without affecting the other(s).

- Pattern Interaction Types.

  Intersection: Intersecting patterns exhibit a “talks to” or “uses a” communication and interaction scheme. We classified a pattern as an intersection type when there was no evidence that it was either a Composite, or an Embedded type.

Table 1. The frequency of occurrences of design patterns in the referenced papers.

Pattern	Pattern Group		
	Creational	Structural	Behavioral
Factory Method	6		
Abstract Factory	7		
Builder	3		
Prototype	1		
Singleton	6		
Adapter		4	
Bridge		2	
Composite		8	
Decorator		2	
Facade		3	
Flyweight		2	
Proxy		5	
Chain of Responsibility			2
Command			4
Interpreter			2
Iterator			3
Mediator			5
Memento			1
Observer			11
State			4
Strategy			8
Template			6
Visitor			4
Totals	23	26	50

Composite: Composite patterns have pattern elements as components. These components are wholly contained in the composite pattern. Composite patterns are recognizable as a "together make up a" relationship — the composite has emergent properties not found in the parts. The composites that we found are not listed as patterns in Gamma et al [8].

Embedded: Embedded patterns represents the interaction best described as a "has a" relationship. The parent pattern includes an instance of the embedded pattern in it. It differs from the composite in that the parent pattern is not a composite of multiple patterns, but has independent content in addition to the embedded pattern. It differs from the intersection "uses a" scheme; the used pattern is entirely within the using pattern's structure.

We found 17 coupled pattern multiples. The identification of coupled patterns was easy in some cases using the descriptions in the examples. In other cases, the reviewer recognized patterns by examining the class structure. Figure 2 shows an example of one of the coupled patterns found in Masuda et al [10]. In this example, an instance of a Template Method pattern is coupled with an instance of a Builder pattern through a shared class. Class Translator plays a role in both patterns. The interaction is of the type "talks with a" which places the patterns into the "Intersection" grouping. Table 6 shows the tabulation of patterns grouped into coupling types.

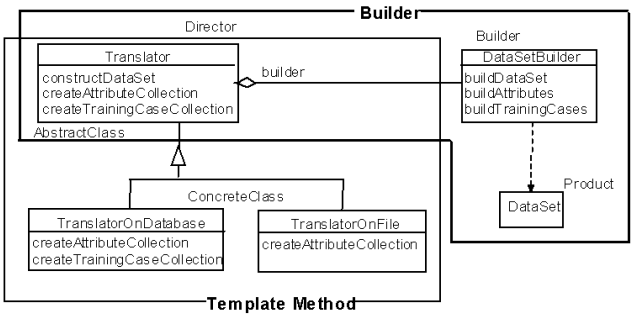

Figure 2. Coupled Patterns in Masuda et al [10].

There are clearly more cases of the intersection category of pattern coupling in the tabulated data. Most of the sets of coupled patterns are intersections.

7. Pattern Coupling and Design Quality

Ultimately, we are concerned with the relationship between pattern use and external quality factors such as maintainability, reusability, and factorability — the ability to restructure a system design.

If we assume that the authors of the papers in our data set had the intention of producing reusable and adaptable designs, then many of our results should have been expected. Embedded patterns are not easily modified without affecting the parent pattern, and modifications of composite patterns affect their "composite parent". The use of an embedded or composite pattern set is best delegated to those application specific needs where there is not likely to be a need for future modification of the component patterns in lieu of modifying the overall parent.

Embedding patterns results in a tendency to treat the parent pattern as the modular unit, with only limited future access to the embedded component. The same is true of composite pattern sets. Large-scale macro pattern languages may make the patterns in the Embedded and Composite categories as easy to use and as flexible as those in the Intersection category. We expect the use of pattern Intersections to continue to predominate in pattern coupling instances, since the need to connect patterns with each other to promote

Table 2. Classification of Pattern Couplings

Coupled Patterns	Subtype			Strength	
	Intersection	Composite	Embedded (parent listed 1st)	Loose	Tight
Strategy–Abstract Factory	1			1	
Visitor–Template	1			1	
Builder–Template-Strategy	1			1	
Abstract Factory–Factory	1			1	
Mediator–Observer	1			1	
Composite–Visitor	1			1	
Factory–Prototype	1			1	
Singleton–State–Command	2				2
Singleton–State–Observer	1				1
Composite–Decorator–Flyweight	1				1
Observer–Facade–Template	1			1	
Bridge–Proxy–Flyweight	1			1	
Facade–Observer	1			1	
Mediator–Strategy	1				1
Adapter–Strategy	1				1
State–Singleton	1				1
Interpreter–Builder–Abstract Factory		1			1
Visitor–Command		1			1
Composite–Proxy		1			1
Observer–Singleton		1			1
Strategy–Observer–Composite			1	1	
Bridge–Observer–Proxy–Abstract Factory–Factory			1	1	
Mediator–Observer–Chain of Responsibility–Composite			1		1
Composite–Interpreter			1		1
Totals	17	4	4	12	13

modularity and factorability will remain and that is what Intersection type couplings can support.

Unfortunately, the use of Intersection type pattern couplings does not guarantee good software designs. Intersections can create both loosely and tightly coupled patterns. A tightly coupled class structure can result in a design that is very hard to break apart and modularize, making the design difficult to modify. Changing one class affects other related classes in a tightly coupled structure.

Our initial assumption was that the research data set would represent "good" uses of design patterns and that most instances of pattern coupling would represent good design practices. At the very least, the research data set represents the most visible examples of coupled patterns. We are most interested in the occurrences of loosely verses tightly coupled pattern sets. As expected, all of the embedded couplings are tightly coupled. The composite couplings are evenly divided between loosely and tightly coupled pattern sets, in part, because, if there were tight couplings between any of the patterns (i.e. one embedded pattern), then we classified the entire pattern set was as tightly coupled. Myers first suggested using the strongest connection to classify instances of coupling [12]. To lower the strength of coupling, eliminate the strongest connections.

We expected a high preponderance of loosely coupled pattern sets among those classified as Intersection couplings. However, seven of the 17 Intersection couplings are tightly coupled, which, with our small sample, includes far more tightly coupled Intersections that we expected. One explanation is that the Singleton pattern is a popular embedded type and its use resulted in a classification of tightly coupled wherever it is applied. Couplings involving the Singleton pattern are notable exceptions to the rule that patterns should be loosely coupled. In many applications the use of a Singleton type pattern can result in code efficiencies due to removal of redundant code (i.e. as a single null node shared for all linked lists in a module). Although instances of the Singleton pattern tend to be tightly coupled to other patterns, it is typically used in an application specific context that developers do not expect to be modified after the first use in a design. Of course, it is difficult to predict what kind of modifications will be needed in an uncertain future.

If we eliminate the effect on our results of the Singleton pattern on the coupling classifications, then the Intersection couplings exhibited loose coupling. Only three of the seven pattern groups that are tightly coupled and exhibit Intersection coupling are tightly coupled for reasons other than the use of the Singleton pattern.

The results of the study tend to bear out the traditional views on designing for software quality. Maintainability,

factorability, and reusability are all supported by modular software with loosely coupled interfaces and abstraction of details to prevent modification of one component from impacting others. The design pattern couplings most often used in the research data set supports this traditional view. Both industry and academic software development papers were in agreement on the value of modular software. The use of patterns not conducive to loose coupling is not eliminated, but rather delegated to specific applications where the impact on modularity is minimized.

8. Limitations

The greatest threat to the validity of our results is a limited data set. First, the composite pattern sets are all from the same data source [15]. We expect that composite pattern couplings will play an increasing role in pattern based design as more and more macro level design structures are developed. Second, we classified coupled pattern sets into the default "Intersection" category if it did not meet the criteria of another category. Finally, we depended on a manual process for classification using one evaluator, the first author. Future research should use an expanded data set, more refined classification criteria, and either an automated classification mechanism or independent expert assessments.

9. Conclusions

We find examples of coupled OO design patterns in theoretical and applied research as well as in industry case studies. The examples are split nearly evenly between sets of tightly coupled patterns and sets of loosely coupled patterns. Sets of coupled patterns classified as exhibiting Composite Coupling and those containing an instance of the Singleton pattern tend to be tightly coupled. However, expanded data is needed to generalize the results to the set of all OO software designs.

Further studies should lead to an understanding of which pattern constructs are effective and lead to more adaptable systems for particular applications. Eventually this work may lead to mechanisms to evaluate macro pattern couplings for new Composite patterns or pattern languages.

Patterns are clearly becoming a popular design mechanism, and as their use increases, there will be a growing need for good pattern coupling methods — if we design a system as a set of patterns, these patterns must communicate. By examining how patterns have been coupled in published examples, we can better understand the benefits and costs of design options involving connected patterns.

10. Acknowledgements

This work is partially supported by U.S. National Science Foundation grant CCR-0098202, and by a grant from the Colorado Advanced Software Institute (CASI). CASI is sponsored in part by the Colorado Commission on Higher Education (CCHE), an agency of the State of Colorado.

References

[1] F. Balaguer, S. Gordillo, and F. Das Neves. Generating the architecture of gis applications with design patterns. *Proc. 5th Int. Workshop on Advances in Geographic Information Systems*, pages 30–34, 1997.

[2] S. Barkataki, S. Harte, and T. Dinh. Reengineering a legacy system using design patterns and Ada-95 object-oriented features. *Proc. ACM Int. Conf. Ada Technology*, page 148, 1998.

[3] M. P. Cline. The pros and cons of adopting and applying design patterns in the real world. *Communications of the ACM*, 39(10):47–49, Oct. 1996.

[4] J. W. Cooper. Using design patterns. *Communications of the ACM*, 41(6):65–68, June 1998.

[5] P. Dagermo and J. Knutsson. Development of an object-oriented framework for vessel control systems. Dover Consortium 96. ftp://st.cs.uiuc.edu/pub/patterns/papers/Dover.ps.

[6] M. Duell, J. Goodsen, and L. Rising. Non-software examples of software design patterns. *OOPSLA'97 Addendum*, pages 120–124, 1997.

[7] R. Engel, H. Hüni, and R. Johnson. A framework for network protocol software. *Proc. OOPSLA'95*, pages 358–369, 1995.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[9] D. B. Lange and Y. Nakamura. Interactive visualization of design patterns can help in framework understanding. *Proc. OOPSLA'95*, pages 342–357, 1995.

[10] G. Masuda, N. Sakamoto, and K. Ushijima. Applying design patterns to decision tree learning system. *Proc. ACM SIGSOFT Int. Symp. Foundations of Software Engineering*, pages 111–120, 1998.

[11] T. Mikkonen. Formalizing design patterns. *Proc. Int. Conf. Software Engineering (ICSE'98*, pages 115–124, 1998.

[12] G. Myers. *Composite/Structural Design*. Van Nostrand Reinhold, 1978.

[13] D. Nguyen. Design patterns for data structures. *Proc. SIGCSE Symp.*, pages 336–340, 1998.

[14] G. Odenthal and K. Quibeldey-Cirkel. Using patterns for design and documentation. *Proc. ECOOP'97*, 1997.

[15] D. Riehle. Composite design patterns. *Proc. OOPSLA'97*, pages 218–228, 1997.

[16] H. A. Schmid. Creating the architecture of a manufacturing framework by design patterns. *Proc. OOPSLA'95*, pages 370–384, 1995.

[17] D. C. Schmidt. Using design patterns to develop reusable object-oriented communication software. *Communications of the ACM*, 38(10):65–74, Oct. 1995.

[18] P. M. Yelland. Creating host compliance in a portable framework: a study in the reuse of design patterns. *Proc. OOPSLA'96*, pages 18–29, 1996.