

Model-based Software Reuse Using Stable Analysis Patterns

Haitham Hamza, Mohamed E. Fayad
Computer Science and Engineering Dept.
University of Nebraska-Lincoln
Lincoln, NE 68588, USA
{hhamza,fayad}@cse.unl.edu

Abstract

The challenge of building efficient reusable software artifacts is the focus of several schools of thought in software engineering. Software analysis patterns are recurring and reusable models. However, there are several deficiencies with analysis patterns. These deficiencies make it difficult to use analysis patterns as efficient reusable artifacts. This paper proposes eight essential properties to evaluate pattern reusability. In addition, the concept of Stability Analysis Patterns is introduced. This paper contrasts stable analysis patterns with some analysis patterns using the proposed properties.

1. Introduction

Since the inception of object-oriented concepts researchers and practitioners alike have held fast to the belief that reuse vastly improves the quality of software products, while simultaneously reducing cost and condensing lifecycles. Many reuse software communities have evolved in recent years, including Aspect-Oriented Programming (AOP), Component-Based Software Engineering community, and many others.

Analysis patterns are conceptual models that model the knowledge domain of the problem. Analysis patterns, as reusable artifacts have been widely heralded by the software engineering community as a major advance over conventional reuse techniques, and rightly so. However, analysis patterns have not realized their full potential. Analysis patterns are insufficiently mature to be considered as a base for building reusable software models. Understanding the cause of this immaturity is the first step in achieving real reuse of analysis patterns.

As models, analysis patterns must satisfy the six basic model properties introduced in [5]. That is, to be simple, complete and most probably accurate, testable, stable, to have visual representation, and to be easily understood. In addition to these six properties, reusable artifacts must

satisfy two additional metrics: first to be general, and second to be easily and actually reused. Thus, a pattern that models a specific problem should be constructed so that it is easily reused whenever the problem occurs, and independent of the context in which the problem appears.

Software stability concepts introduced in [1] have demonstrated great promise in the area of software reuse and lifecycle improvement. Software stability models apply the concepts of “Enduring Business Themes” (EBTs) and “Business Objects” (BOs). These concepts have been shown to produce models that are both stable over time, and stable across various paradigm shifts within a domain or application context. By applying stability model concepts to the notion of analysis patterns we propose the concept of *Stable Analysis Patterns*. The idea behind the stable analysis patterns is to analyze the problem under consideration in terms of its EBTs and BOs with the goal of increased stability and broader reuse.

In the remainder of this paper we will introduce the eight essential properties of analysis patterns (Section 2), and discuss the different groups approaches for building analysis patterns (Section 3). We will study some example patterns reflecting each of the aforementioned groups (Section 4), and compare these approaches (Section 5). Conclusions are presented in Section 6.

2. Essential Properties of Analysis Patterns

In this section we examine the eight properties of efficient reusable models. Satisfying these eight properties does not guarantee an efficient reusable model; however, in practice, lacking any of these properties will affect the reusability of the model. These eight essential properties are:

1. Simple: a pattern is not intended to represent a model for a complete system; rather it models a specific problem that commonly appears within larger systems. Systems, by their nature, combine many problems. Thus, they are modeled using a collection of analysis patterns. In fact,

each analysis pattern should focus on one specific problem; otherwise, many problems arise. Without decomposing a system into components, models become unreasonably complex, the generality of the patterns are adversely affected, and the model becomes highly non-intuitive. If a pattern is used to model an overly broad portion of a system, the generality of resulting patterns is sacrificed – the maxim holds: the probability of the occurrence of all the problems together is less than the probability of the occurrence of each problem individually. For example, modeling the “payment” problem with “buying a car” is not effective since the “payment” problem may appear in unlimited number of problems. Pattern completeness is also sacrificed when we model a system at an improper level of resolution, because the analyst’s focus is not on a specific problem, and it is likely that important feature of the system and its subcomponents will be overlooked.

2. Complete and most likely accurate: closely related to the concept of simplicity, this property guarantees that all the required information is present. In order to be considered complete the model should not omit any component. The model must be able to express the essential concepts of its properties. For example, trying to model the whole rental system of any property will force us to miss some of the parts of this system. Renting a car will involve something related to its insurance; however, renting a book from a library has nothing to do with the insurance problem. As a result, pattern that models the rental system, besides lacking the simplicity property, it will not be complete or accurate.

3. Testable: for the model to be testable, it must be specific and unambiguous. Thus, all the classes and the relationships of the model could be qualified and validated.

4. Stable: stability influences the reusability of a model. Stable models are easily adapted and modified without the risk of obsolescence.

5. Graphical or visual: conceptual models are difficult to visualize. Therefore, having a graphical representation for the model aids understanding it.

6. Easy to understand: Conceptual models are complex as they represent a high level of abstraction. Therefore, it is required for analysis patterns to be well described such that they aid in communicating an understanding of the system. Otherwise use of the pattern is neither attractive nor effective.

7. General: This property is essential to ensure model reusability. Pattern models lacking generality become useless, since analysts will tend to build new models rather than spending time and effort to adapt an unruly pattern to fit into an application. Generality means that a pattern that models a specific problem is easily used to model the same problem independent of context. Pattern generality may be divided into two categories: Patterns

that solve problems that frequently appear in different contexts (domain-less patterns), and patterns that solve problems that frequently appear within specific contexts (domain-specific patterns). In the latter sense, the pattern is still considered to be general even if it is only applicable in a certain domain, but in this case, we should make sure that the problem that this pattern models does not occur in other contexts.

8. Easy to use and reuse: analysis patterns should be presented in a clear way that makes them easily reused. It is important to remember that patterns are consumed in larger models. Patterns that are easy to use and designed for reuse stand a greater chance of actually being reused.

3. Classification of Analysis Pattern

One possible classification for analysis patterns is based on the construction approach. Generally, different building approaches categorize analysis patterns into three groups:

Group I: People in this group use their experience to build analysis patterns. Simply, patterns are produced during the course of specific projects. Since no one can be an expert in all fields, domain experts often produce domain specific patterns, even if the problem modeled occurs in many other contexts. People in this category believe that it is unsafe to further abstract patterns generated within certain projects in order to make them reusable in other contexts. They argue that the patterns resulted from extended debate and that the patterns have been tested and validated in the project. Therefore, there is no guarantee that these patterns will be successfully reused in other contexts.

Group II: People in this group use analogy to build their analysis patterns. According to this group, patterns that model complete systems in one context are reused by making an analogy between the pattern and the new application. Thus, by analogy, they change the names of the pattern’s classes to be relevant to the new application. It is also possible to remove or add few classes to the pattern’s model. Even though this group believes that analysis patterns should be built in a way that makes them reused to model the same problem regardless of its context. However, the way they choose to approach this goal makes them end up building templates rather than building patterns.

Group III, which is our group, our approach is based on the software stability concepts [2]. By analyzing the problem in terms of its EBTs and the BOs, the resultant pattern models the core knowledge of the problem. The goal of this approach is stability. As a result, these stable patterns could be used to model the same problem regardless its context.

4. Evaluation of Analysis Patterns Groups

In order to fairly compare the effectiveness of the patterns generated by the three different groups described in the previous section, a pattern that reflects the approach of each group will be examined against the essential properties mentioned earlier in section 2.

4.1 Group I

The *Account Pattern* provided by Fowler [5] is representative of this group. Figure 1 shows the class diagram of the *Account Pattern*. The purpose of this pattern is to provide a model for the “account” problem; thus, we can use this pattern to model banking account for instance. In fact, it was not long time ago when word account has been merely used to indicate banking and financial accounts. Today, the word account alone becomes a vague concept if it is not allied with a word related to a certain context. For instance, besides all the traditional well-known business and banking accounts, today we have e-mail accounts, on-line shopping accounts, on-line learning accounts, subscription account, and many others. As a result, using words such as balance and withdrawal while modeling the account, makes the use of the pattern to model accounts in different contexts, time and effort-consuming, if not, impossible. For instance, suppose we want to model an e-mail account using Martin’s pattern, perhaps the most obvious changes are all the classes’ behaviors, which are completely irrelevant to the email application.

From the simplicity point of view, Fowler’s *Account Pattern* is not considered simple in the sense that it models two different problems at the same time. The first problem is the “account” problem and the second problem is the “entry” problem. In fact, these are two independent problems. Even though they used to appear together in many contexts; however, there is a possibility of having entries without an account, or having an account without entries. As a result, the generality of the pattern is limited. These factors contribute negatively to the reusability of the pattern.

Fowler’s pattern is not complete in the sense that it lacks some of the “basic” concepts that appear frequently in banking accounts. For instance, suppose that we need to use this pattern to model a banking account. In banking accounts it is possible that two or more persons may be holders of the same account. Perhaps, there is a primary holder that has the full authorization to manage and control the account, while each of the other holders will have specific privileges for using the same account. Such situation cannot be handled while using Martin’s account. Thus, Martin’s pattern is not applicable to some of the usual financial and banking account situations. Since

significant effort is required to adapt this pattern to other circumstances, the stability of the system is limited and the pattern structure is not stable over time.

This pattern is graphical in the sense that it has a graphical model that describes it (the class diagram). Having such a graphical representation will make the pattern visually testable.



Figure 1. Martin’s Account pattern [5]

4.2 Group II

Figure 2 provides the class diagram of the *Resource Rental Pattern* [6]. The objective of the pattern is to provide a model that could be reused to model the problem of renting any resource. Figure 3 provides an example of the *Resource Rental Pattern* applied in the context of a library service [6]. Many examples for applying this pattern to different applications are suggested in [6].

This pattern models a complete resource rental system; thus, it models a collection of problems, whereas each of these problems could be modeled individually. For example, the “payment” process is a stand-alone problem, which could appear in many other contexts. Therefore, having a pattern that models the “payment” problem alone will be more effective since such pattern will be reused in many other applications.

The resource rental pattern lacks the simplicity. In addition, it is not general, because it is not applicable to any resource rental. One of the most basic steps in the automobile rental process is the question of insurance. There is nothing in this model that can be used to address insurance programs.

Another issue that is not addressed in this pattern, yet it is essential in many renting systems, is the verification process. In many cases, renting a resource might require a membership (as in the case of the universities library) or other identification (such as the driver license in the case of renting a car). There is a link between the completeness of the model and its stability. Reuse is challenging when applying incomplete patterns because many new classes are needed to complete the model.

In our example, suppose we need to model a car rental system using this pattern. Now we have reached the conclusion that we need to add the verification process classes and the insurance process classes. Substantial analysis is needed to complete this new model, hence the

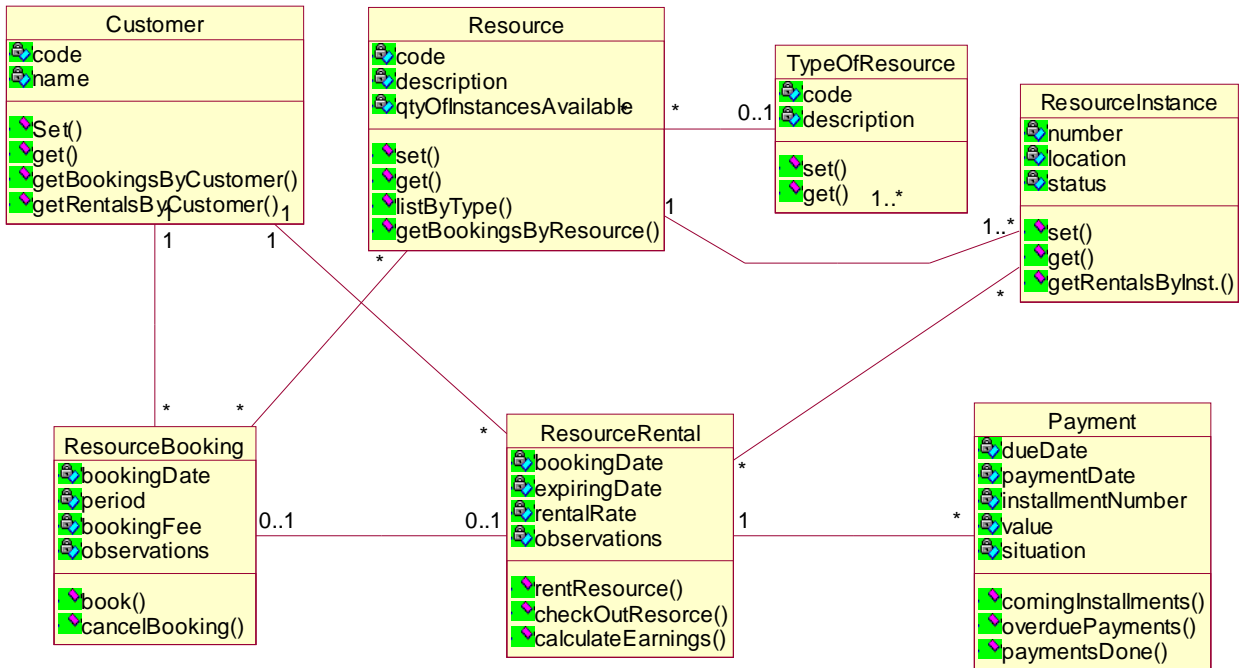


Figure 2. Resource rental pattern

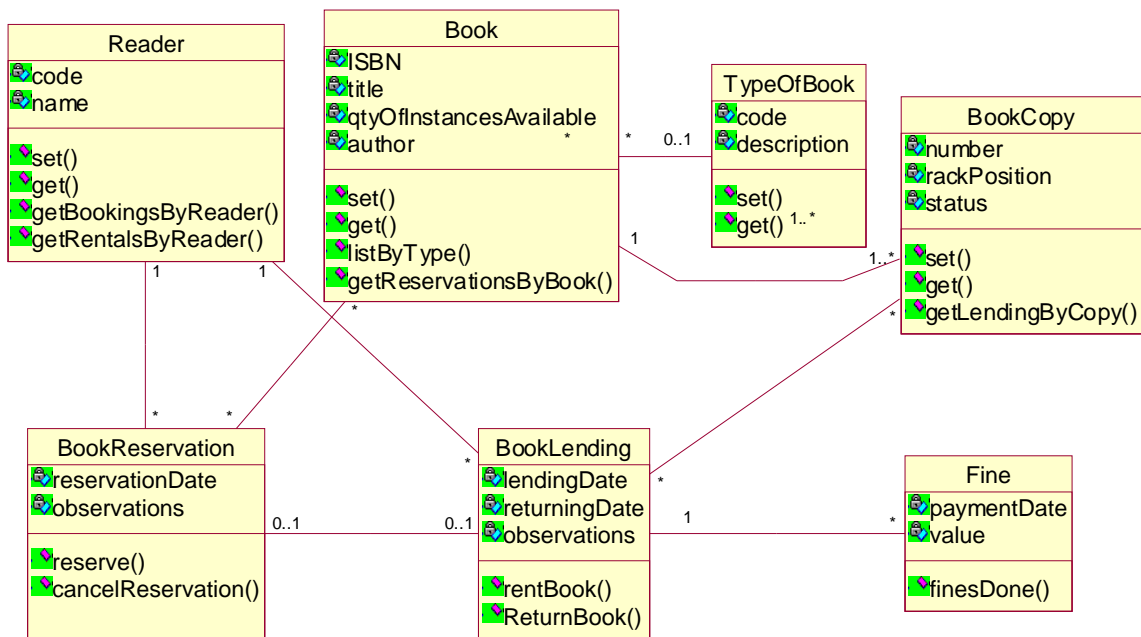


Figure 3. Instantiation of resource rental pattern for a library service [6]

analyst may be inclined to build a new model from scratch. Therefore, the pattern will not be reused. As in the first group, the existence of a class diagram to represent the model makes the model graphical. As a result the pattern is readily tested.

4.3 Group III

This group reflects our proposed solution for building reusable analysis models and avoiding most of the common problems we found in the other groups. In order to make the evaluation of this group patterns more interesting, the pattern example that is chosen is the stability version of the *Account* pattern introduced by Fowler.

From the stability point of view, the model that focuses on the account problem has nothing to do with the entry problem. Thus, it is required to have different patterns for each individual problem. In this manner, the simplicity of our models is guaranteed since each pattern will focus on a specific problem.

Stability goes further by providing other classes that do not exist in Fowler's model. Figure 4 shows the proposed analysis pattern "*AnyAccount*" that provides the stable model for the "account" problem. The new classes that appear in the stability model help us to handle those circumstances that Fowler's model fails to cover; thus, the model becomes accurate and complete. For instance, the use of the EBT of "Ownership" and the BO of "Holder" in the modeling of the account aids in contexts where there is a difference between the account owner, and those who are authorized to use the account under certain rules, should made clear.

"Ownership" is an enduring concept, which will never change independent of context. On the other hand, the "Holder" here is externally stable and never change with time, although the holders of the account could internally change (holder may get ill, for example); however, they are still the holder of the account. As we can see in the pattern class diagram, the inherited objects from the "Holder" object model the different roles of the different levels of the usability of that account. This pattern's structure is stable over the time and general enough to handle different applications that involve accounts and the different situations within the same application as well.

On the other hand, thinking about "entry" as a stand-alone problem forces us to build a pattern that models any entry regardless of context. Using the stability concepts we were able to come up with a stable pattern the models any entry for any application. This pattern is called "*AnyEntry*" pattern and its class diagram is given in Figure 5.

By combining the pattern that models the account problem (the "*AnyAccount*" pattern) with that which models the entry problem (the "*AnyEntry*" pattern) we can

demonstrate the ease of reusing stability models to construct comprehensive models. Figure 6 shows the class diagram for this third pattern. The "*AccountWithEntry*" pattern could be used to model any account that has entries associated with it, as in the case of banking accounts and email accounts for example.

5. Comparison of Analysis Patterns Groups

Based on the essential properties discussed in section 2, table 1 summarizes the results of the three analysis pattern groups.

6. Conclusion

Analysis patterns could form a foundation for building reusable software assets. However, this evaluation of some analysis patterns show that these patterns lack many essential properties. As a result, their reusability is diminished. Software stability has been proposed as a solution for the deficiencies encountered in analysis patterns. *Stable analysis patterns* demonstrate effectiveness by satisfying all the proposed properties. Therefore, the application of stable analysis patterns is a promising approach meriting further research among the reuse communities.

7. References

- [1] M. E. Fayad, A. Altman, "Introduction to Software Stability", *Communications of the ACM*, Vol. 44, No. 9, September 2001.
- [2] M. E. Fayad, "*Accomplishing Software Stability*", *Communications of the ACM*, Vol. 45, No. 1, January 2002.
- [3] M. E. Fayad, "*How to Deal with Software Stability*", *Communications of the ACM*, Vol. 45, No. 4, April 2002.
- [4] M. E. Fayad and M. Laitinen, "Transition to Object-Oriented Software Developments", New York: Wiley & Sons, August 1998.
- [5] M. Fowler, "Analysis Patterns: Reusable Object Models", Addison-Wesley, 1997.
- [6] R. T. Vaccare Braga et. al., "A Confederation of Patterns for Business resource Management" *Proceedings of Pattern Language of Programs '98 (PLOP'98)*, 1998.

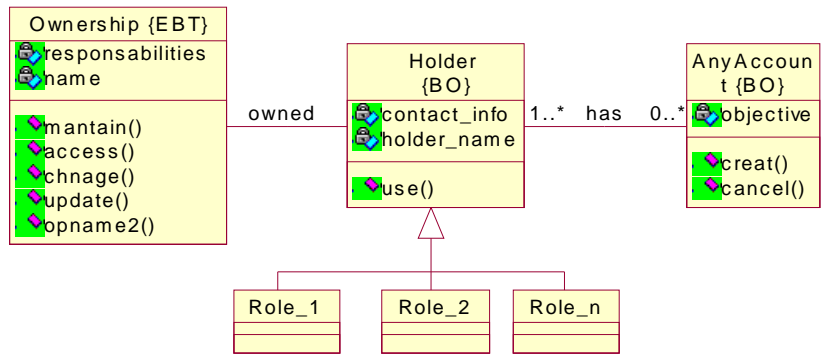


Figure 4. *AnyAccount* pattern class diagram

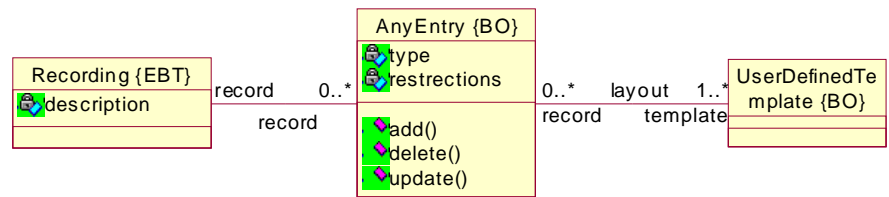


Figure 5. *AnyEntry* pattern class diagram

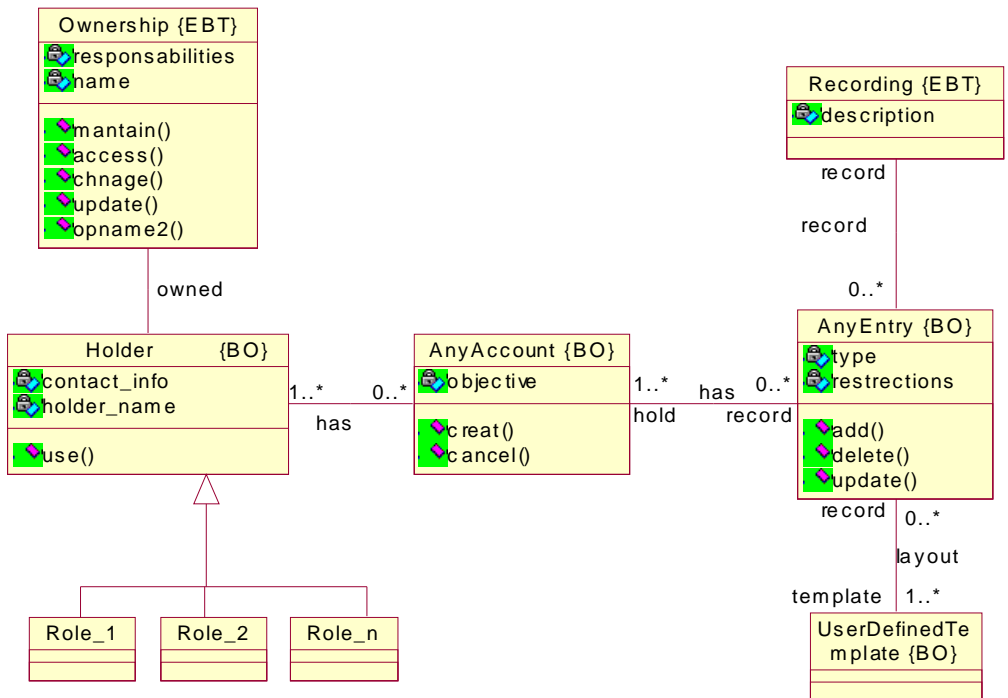


Figure 6. *AccountWithEntry* pattern

Table 1. Comparison of analysis patterns groups

Properties	Evaluation		
	Group I	Group II	Our Group
Simple	No. It models two different problems.	No. It models an entire system.	Yes. Each model focuses on one problem.
Complete and most likely accurate	No. It does not cover all the circumstances that might occur in the application.	No. It is not sufficiently general to address the requirements of different renting applications.	Yes. Because each model focuses on a specific problem, all situations within the problem can be easily covered.
Stable	No. This pattern cannot model all types of today's accounts. Thus, we will always need to do major changes to reuse this pattern for different applications.	No. Using this pattern to model different application will need major changes. For instance, adding the verification process to the model will need a lot of changes.	Yes. The patterns in this group are built with stability in mind. The use of EBTs and BOs, ensure stability in the model.
Testable	Yes. Since the pattern can be visualized; thus, we can, at least, visually test it.	Yes. Since the pattern can be visualized; thus, we can, at least, visually test it.	Yes. Since the pattern can be visualized; thus, we can, at least, visually test it.
Easy to understand	Yes. Generally speaking, despite the accuracy of the pattern, it is easy to understand its structure.	Yes. Despite the accuracy of the pattern, it is easy to understand its structure.	Yes. Since the used EBTs and the BOs reflect the concepts that we are familiar with; it is easy to understand the model structure.
Graphical or visual	Yes. The pattern has a graphical presentation, which is the class diagram.	Yes. The pattern has a graphical presentation, which is the class diagram.	Yes. The pattern has a graphical presentation, which is the class diagram.
General	No. We cannot use it to model the account in other contexts other than monetary application. Also, it dose not cover the cases of having accounts without entries and vise versa. Moreover, the pattern dose not cover some of the situations such as having more than one holder for the same account.	No. This pattern cannot be used to model the rental of some resources. For instance car rental, since there is nothing in the model that cover the insurance issues, which is an essential part of any cat rental process.	Yes. Because of the stability concept, our models focus in a specific problem trying to flush the core knowledge underneath the surface of the problem. Since the core knowledge of any problem is constant regardless the context that this problem might appear in, the model of the problem is general and can apply to the problem whenever it occur.
Easy to use and reuse	No. Using the patterns of this group in different application that they originally built for, if it possible is not an easy task.	No. As we mentioned before, we will need to do major changes to use this pattern in different application such as for car renting.	Yes. Using the pattern by itself or the integration of few patterns are both easy to be done. This property is demonstrated by introducing the third pattern shown in figure 6.