The Impact of Stability on Design Patterns Implementation

Shasha Wu, Ahmed Mahdy, and Mohamed E. Fayad
Computer Science and Engineering Dept.
University of Nebraska-Lincoln
Lincoln, NE 68588, USA
{shwu, amahdy, fayad}@cse.unl.edu

Abstract

Design patterns are reusable constructs. They are stable and adaptable by definition. Unfortunately, in order to achieve usability, the elegant characteristics displayed in design patterns, such as stability, adaptability and generality are diminished in the implemented models. Thus, a discrepancy is revealed between the design patterns and these models. This paper suggests the use of Software Stability [6] as a solution for resolving the inconsistency between the design patterns and their implementation.

1. Context

The basic message of this paper is that the implementation of design patterns leaves the programmer with no trace of what patterns were applied and what design decisions were made. Thus, when changes have to be made, the entire design has to be almost entirely reconstructed.

We will explain this with an analogy to driving rules. The pattern instances correspond to actual driving and the problem context can be thought of as different driving situations. The rules are always constant although the actual driving may change under different circumstances. To drive safely and efficiently, people must learn the rules first and then apply them according to the different circumstances. Every time they drive, they must keep the rules in mind in order to adapt to different situations. That is the key to stable driving. Expecting these models to be stable under changes is similar to applying past driving sequence actions to different times and locations.

2. Problem

"A pattern is a plan, rather than a specific implementation." [7]. They are "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context" [4]. Consequently, design patterns must be stable, abstract, and common. This makes them unsuitable for detailed implementation on specific problems. "The common practice, design patterns are used only during the initial, conceptual design." [5] They guide the modeling process and detailed design process. By instantiation, the original abstract objects of design patterns are replaced with concrete objects representing instances of a specific domain; plus other objects. Those instances

are the actual guide for programming. Figure 1 shows the mechanism of the adaptation process of a design pattern.

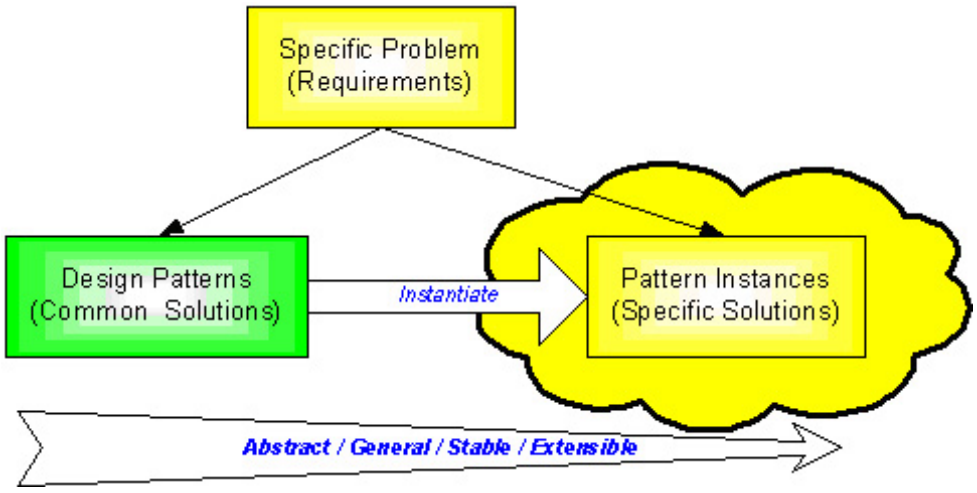

Figure 1. Current Approach of Implementing Design Patterns

In current approaches of implementing design patterns, design patterns and their instances are separate models: abstract and concrete domains. After instantiation, the instances are less abstract, less stable and difficult to extend, compared to their design patterns. This is because a pattern gains its quality factors from being a recurrent design issue and solution in multiple contexts/domains. Design patterns are not traceable from the resulting implementation model. Thus, the solution loses an important quality of patterns: stability in the face of changes.

The instantiation process can be used to solve this discrepancy. But the description of this process is not traceable from the resulting implementation model too. This means a specific piece of code (implementation model) implementing a given solution, lacks the ability to trace back to its blueprint design pattern. This disallows the extension of the model for future changes. Thus, the theme of this paper is that the pattern itself is lost in the implementation. Is there a way to convey the pattern characteristics to its instances to achieve both usability and stability in the resulting models?

3. Solution

One solution to this problem is proposed in [5]. It suggests using Pattern Classes to improve the readability and maintainability of final code.This paper suggests the use of stable models and Enduring Business Themes [6] to help restore what would otherwise be lost in the implementation. The model loses its generality and abstraction after instantiation, causing it to be weak in adaptability and extensibility. When later developers want to extend the software, they cannot directly reuse this model because the

implementation of pattern does not allow tracing back to the abstract design pattern. They have to reconstruct the whole model from the original design patterns.

The Software Stability approach [6] has the potential to build such models. "A Software Stability Model (SSM) can be triply partitioned into levels: Enduring Business Themes (EBTs), Business Objects (BOs), and Industrial Objects (IOs). EBTs represent intangible objects that remain stable internally and externally. BOs are objects that are internally adaptable but externally stable, and IOs are the external interface of the system. In addition to the conceptual differences between EBTs and BOs, a BO can be distinguished from an EBT by tangibility. While EBTs are completely intangible concepts, BOs are partially tangible. These artifacts develop a hierarchal order for the system objects, from totally stable at the EBTs level to unstable at the IOs level, through adaptable though stable at the BOs level. The stable objects of the system are those do not change with time" [1]. From an abstractness aspect, the EBTs are completely abstract, the BOs are mostly abstract, and the IOs are not abstract. Hence, the EBTs and BOs are common among applications with similar core, while the IOs are those object differentiate an application from another. Figure 2 shows the SSM structure.

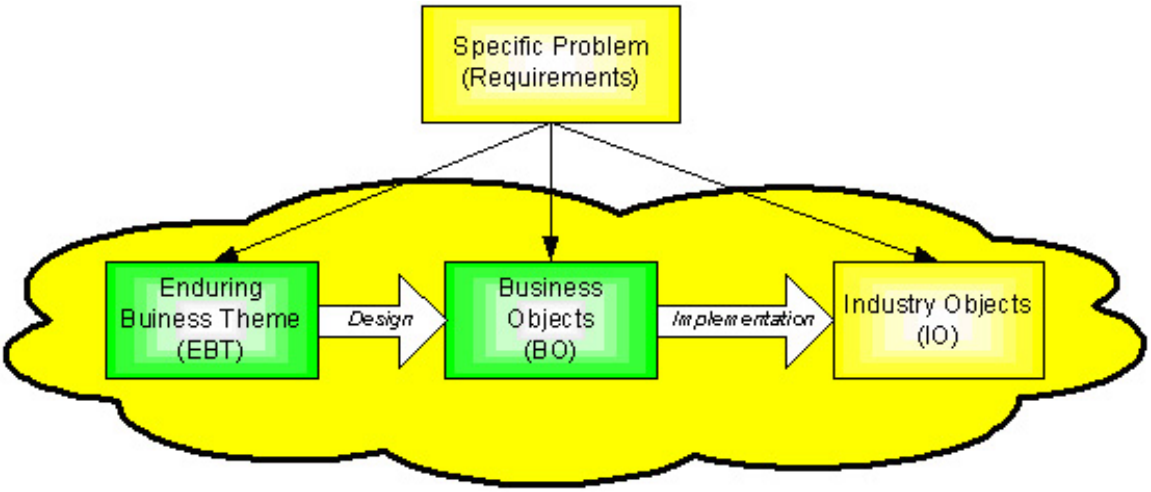

Figure 2. Stability Model Structure

The EBTs and BOs are abstract like design patterns, but they do not disappear in the implementation. They describe a common solution to the problem. The IOs are the same as the pattern instance. They are concrete, problem-specific and unstable. When we include the original design patterns in the final instance models and provide their collaborations, the model bears a striking resemblance to the SSM. By associating these two parts in the SSM instead of separating them as in the current Design patterns approach, we add a stable core to the resulting model. This keeps the model stable over changes.

Future developers can trace the ideas of the original model designers and extend the model safely, as the core remains stable. Combining the abstraction and generalization from the original design patterns (i.e. EBTs and BOs) and the specification of the instances (i.e. IOs), the SSM achieves stability and usability concurrently. It shoots two birds with one stone.

4. Example

Figure 3 shows a design pattern on sales transactions [7].

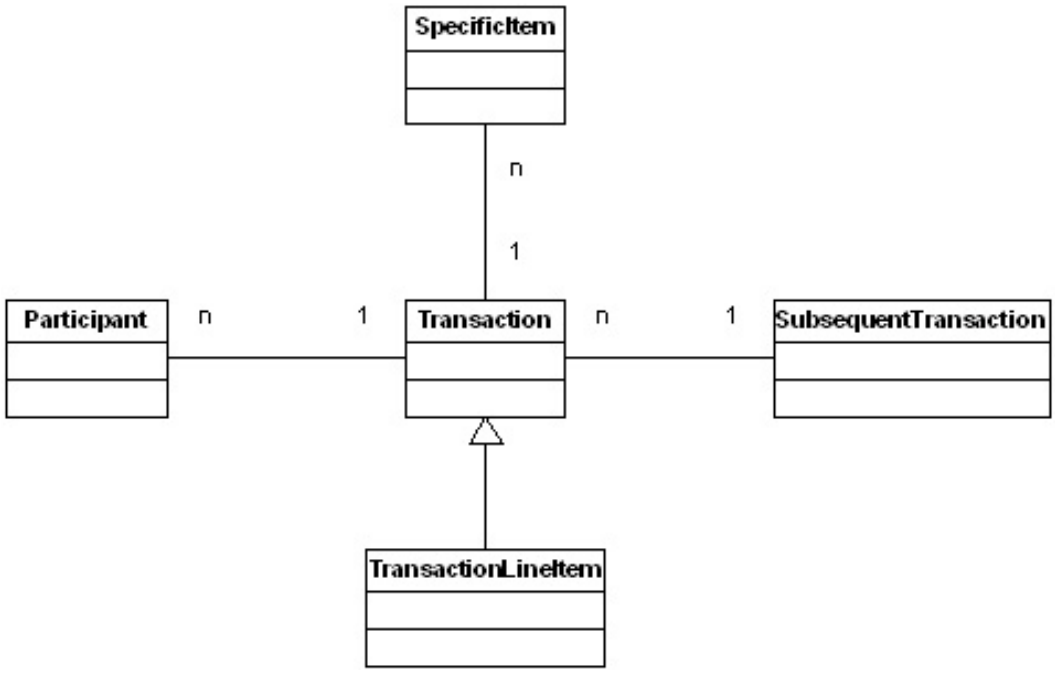

Figure 3. Design Patterns for Transaction (Coad 1995 [7]. Figure 6-32)

Figure 4 shows an instance of the transaction pattern [7].

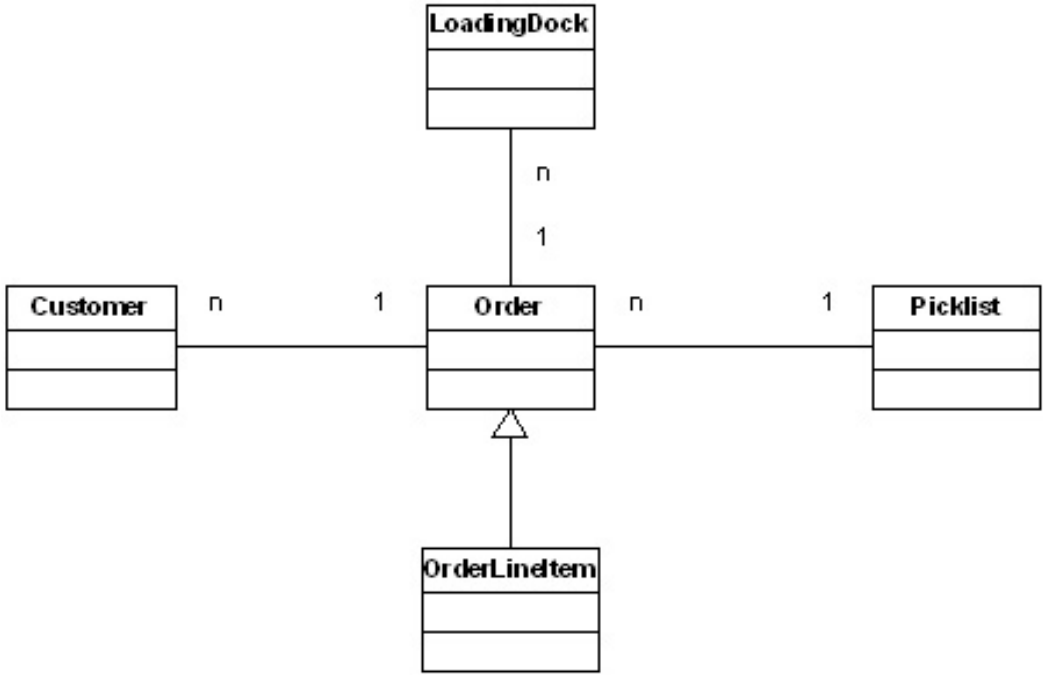

Figure 4. Design Patterns Instances on order (Coad 1995 [7]. Figure 6-33)

There is no instantiation processes shown in Figure 4; the final result of model using the current design patterns implementation approach. The instances were deduced from the design patterns, but the design patterns were discarded in the final model. From Figure 3, it is very difficult and uncertain to induct "SubsequentTransaction" from "Picklist". Thus a designer cannot go back to the original designs to extend existing models. Each time that the designers want to extend the models, they must go all the way back to the original design patterns and reconstruct the whole model. For example, if a new object named "Planlist" is introduced in another situation, the information displayed in Figure 4 is not sufficient to define the new "Planlist" object.

Figure 5 describes the same problem as Figure 4 using the Software Stability approach. Using this new model, we can easily change the IOs without worrying about destroying the whole structure of the model. Suppose for example, the circumstances change and we need to introduce "Planlist" to satisfy a new requirement. Using the SSM we can easily instantiate "Planlist" from "SubsequentTransaction" as an IO to extend the model without modifying the whole structure. The EBTs and BOs remain constant during this process. This efficiently keeps the core structure and design ideas of this model unchanged over time.

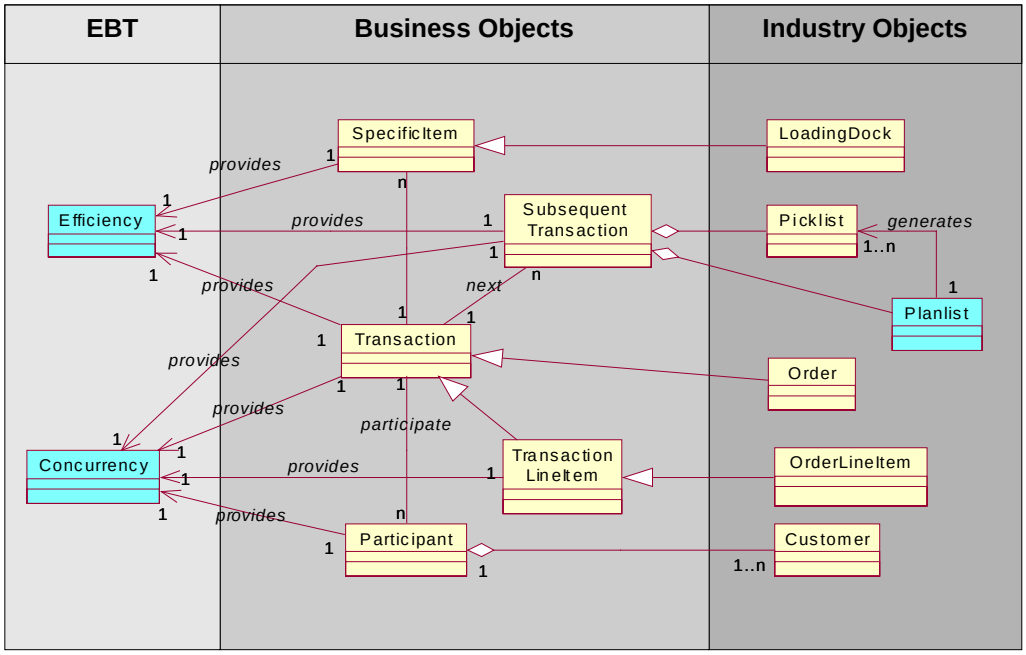

Figure 5. Stability model for Transaction Design Patterns on order instance

5. Conclusion

"Patterns explicitly capture expert knowledge and design tradeoffs, and make this expertise more widely available" [3]. However, the implementation of design patterns has difficulty constructing stable software products because much of the design abstractions are lost in the implementation, with no traceability back to the design patterns to accommodate new variations. A SSM has the ability to extend its usage of pattern implementation without modifying its whole structure.

The software stability approach provides a practical method of explicitly describing the two-way mapping relationship between design patterns and their implementations/instances. The EBTs and BOs represent the core of the model and are constant under change. This allows the model to remain stable. The IOs can be modified easily and safely according to the specific problem and their original design patterns (i.e. EBTs and BOs). The abstract parts (EBTs and BOs) and concrete parts (IOs) are separated clearly but connected closely in the SSM. Therefore, designers have a means of tracing back and re-executing the instantiation process to extend the model. Accordingly, the stability of the model becomes feasible.

Acknowledgement

We would like to thank Ali Arsanjani for shepherding and putting this paper in the final shape.

References

[1] A. Mahdy, M.E. Fayad, H. Hamza, and P. Tugnawat, "*Stable and Reusable Model-Based Architectures*", 12th Workshop on Model-based Software Reuse, 16th ECOOP 2002, Malaga, Spain.

[2] D.C. Schmidt, M.E. Fayad, and R. Johnson, "*Software Patterns*", The Special Issues in Communications of the ACM, Vol. 39, No. 10, October 1996.

[3] D.L. Levine and D.C. Schmidt, "*Introduction to Patterns and Frameworks*", Department of Computer, Science Washington University, St. Louis. Available at: http://www.cs.wustl.edu/~schmidt/PDF/patterns-intro4.pdf.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "*Design Patterns: elements of reusable object-oriented software*". Addison-Wesley, 1995.

[5] J. Soukup, "*Implementing Patterns*", Available at: http://www.codefarms.com/publications/papers/patterns.html.

[6] M.E. Fayad and A. Altman, "*An Introduction to Software Stability*", Communications of the ACM, Vol. 44. No. 9, September 2001.

[7] P. Coad, D. North, and M. Mayfield, "*Object Models – Strategies, Patterns, & Applications*", Yourdon Press, Prentice-Hall, Inc. New Jersey. 1995.