Multi-Design Application Frameworks

Stefan Hanenberg
Department of Mathematics & Computer Science
University of Essen
Schützenbahn 70, 45117 Essen, Germany
Email shanenbe@cs.uni-essen.de
URL: http://www.cs.uni-essen.de/dawis/shanenbe/

Keywords: Application Frameworks, Aspect-Orientation, Design Patterns
Classification: 1 year&acute;s work

1 Introduction

Application frameworks allow to reuse design and implementation of software. Although they are meanwhile an essential basis for the rapid implementation of large software systems, the reuse of design is quite restrictive. Multi-design application frameworks are combinations of object-oriented and aspect-oriented frameworks, which allow a more flexible reuse by separating design as a concern.

2 Application Frameworks

Application frameworks are meanwhile an essential basis for the rapid implementation of large information systems. They contain domain knowledge that has to be adopted by the application developer. By using a framework developers reuse its design and implementation.
The flexible artifacts within a framework that can or even must be adapted by developers are called *hot spots* [Pre95], those that can't or shouldn't be changed are *frozen spots*. Usually the hot spots are abstract classes or interfaces which must be extended or implemented by the application developer. The main essence of frameworks is, that hot spots are invoked from within the framework itself (cf. [JoF88]) using certain design patterns like strategy, template method etc. (cf. [Gam+95]).
In fact application frameworks not only contain an abstract design and implementation of a domain, they also contain numerous concerns [Dij76] which influenced the framework's design and its functionality. For example a framework based on Enterprise JavaBeans contains numerous aspects [Blv98] and therefore can not be used for client-sided applications, even though it may depict the needed domain.

3 Aspect-Orientation

Aspect-oriented programming (AOP) [Kic+97] extends the object-oriented paradigm by introducing aspects as a new model element. Aspects are elements to express concerns, which are able to crosscut the structure of another model [CzE00]. Software systems can be developed using object-oriented programming languages and extended by binding aspects to them. Within the aspect-oriented terminology the oo-programs are called *primary structure*, the process of binding aspects is called *weaving*. The points at which an aspect crosscuts another model element are called *join points*, the references to join points are called *pointcuts.*

4 Aspect Frameworks

Aspect frameworks are collections of abstract and concrete aspects. Like object-oriented frameworks they also contain hooks and templates, but in addition to methods, abstract pointcuts may be overwritten. That means although certain aspects are already implemented, it is not fixed which elements of a primary structure they crosscut. So abstract aspects allow to reuse the concern they depict for numerous primary structures.

5 Separating Design Patterns as Concern

Design patterns become lost on implementation level within object-oriented applications and frameworks [Sou95]. If programming languages supply implementations of some patterns (e.g. class `Observable` and interface `Observer` in Java) then these patterns become part of the inheritance structure of some classes and there is no way to extend them without the included patterns. It developers decide to implement patterns on their own, it is hardly possible to identify these patterns in the model of an application. If would be more desirable to identify patterns on model level and reuse the patterns' implementations. This can be done using aspect-oriented techniques.

In [HFU00] we showed, how certain design patterns can be encapsulated using aspects and introduced our aspect framework *sideFrame* [Side00] based on the aspect language aspectJ (cf. [LoK98]). This white box aspect framework contains pre-build design patterns which can be weaved to an existing primary structure by overriding abstract methods and pointcuts. The main benefit is, that by weaving design-aspects the implementations of patterns can be reused and on model level these patterns can be identified. Additional, these patterns can be weaved whenever needed without staining the inheritance structure of the participating classes.

By encapsulating design patterns into aspects there is no need any longer to keep certain design patterns within an object-oriented framework when not strictly prescribed by the domain. Instead object-oriented frameworks should additionally be supplied with an aspect framework which allows developers to add specific patterns. The main consequence of this approach is, that object-oriented frameworks become more flexible, because application developer decide which patterns to use in what situations. Another effect of separating design patterns as concerns is, that documenting frameworks becomes more easier of the fact, that interfaces within the framework become more understandable.

6 Example

We assume a quite simple financial framework in Java, which supplies two (concrete) classes `Account` and `Booking`. `Account` has two attributes `owner` and `id` for the owner's name and the account number. A booking just consists of an `amount`, which is in this simple example of type float (figure 1a).

There may be two different kinds of customers for this framework:

1. developers writing client-applications for managing accounts on a single machine
2. developers writing internet applications for managing account via internet

The first type of customer writes a GUIs which needs to observe `Account`- and `Booking`-instances, the other does not. For the framework provider it means either to supply the observable-functionality within the framework and to satisfy the first type's needs (figure 1b), or to neglect it and to offer simple interfaces appropriate for the second type of customer. If the observer-property is neglected, the first customer has to adopt it (figure 1c) by implementing the observer-pattern on his own.

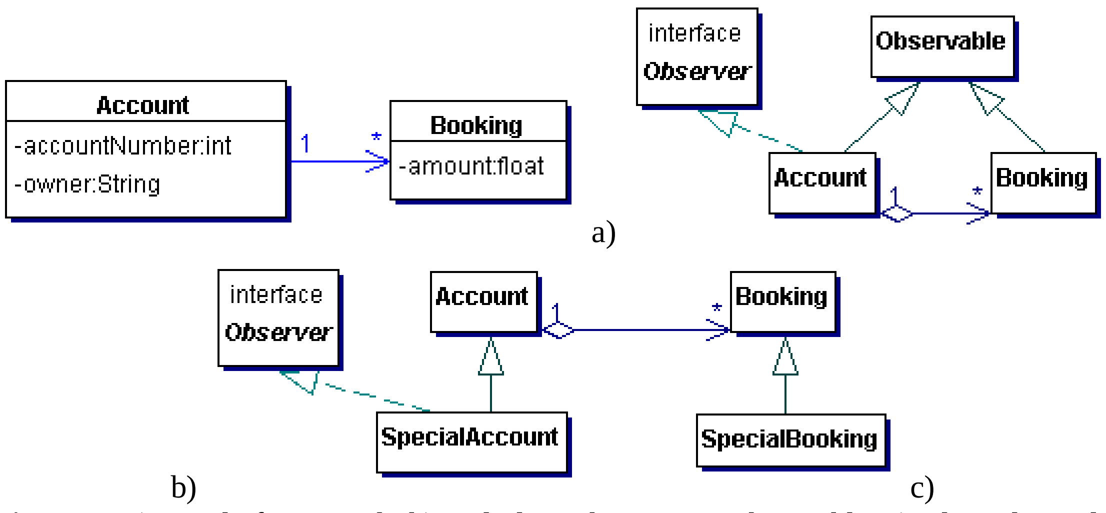

Figure 1: a) simple framework, b) including oberserver / observable, c) adopted simple framework

When separating design concerns within an aspect framework and supplying it in addition, the framework can satisfy both customers' needs. Adding observable- and observer-properties to `Booking` and `Account` means in sideframe to extend the generic aspects of the participants within the observer pattern (`GenericSubject` and `GenericObserver`) and set `Account` and `Booking` as join points (figure 2). The generic aspects contain the needed behavior and interaction of the pattern like attaching the observer to the subject and informing the observer whenever the subject's state changes. So the framework's design can be adopted to the developers' needs.

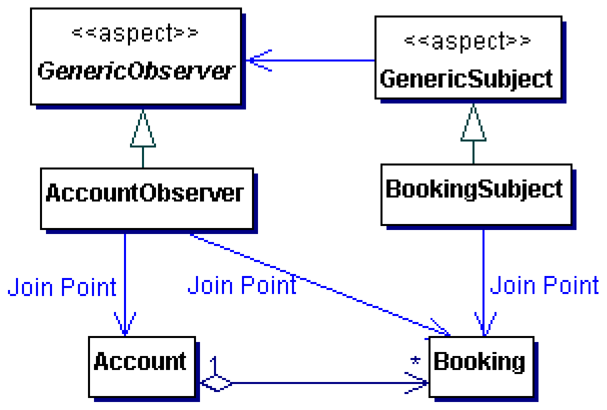

Figure 2: Adopting simple Multi-Design Framework

7 Future Works

In the future we plan to analyze how the structure of aspect-oriented frameworks can be described with well known techniques like metapatterns. In addition we plan to extend sideframe with architectural patterns.

References

[Blv98] Gregory Blank, Gene Vayngrib, *Aspects of Enterprise Java Beans*, Aspect-Oriented Programming (AOP) Workshop at ECOOP'98. 1998

[CzE00] Czarnecki, Krzysztof und Ulrich W. Eisenecker: *Generative Programming: Methods, Tools, and Application*, Addison-Wesley, 2000

[Dij76] Dijkstra, Edsger W.: *A Discipline of Programming.* Prentice-Hall. 1976
[Gam+95] Gamma, Erich, Helm, Richard, Johnson, Ralph E., Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley. 1995.
[HFU00] Hanenberg, Stefan, Franczyk, Bogdan, Unland, Rainer, *Aspect Frameworks: How aspect-orientation an smoothly support the evolution process of software*, submitted
[JoF88] Johnson, Ralph E. , Foote, Brian , Designing reusable classes. Journal of Object-Oriented Programming, 1(5), 1988, pp. 22-35
[Kic+97] Kiczales, Gregor, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier und John Irwin: *Aspect-Oriented Programming.* In: Aksit, Mehmet (Hg.): Proceedings of ECOOP'97. Springer-Verlag. 1997
[LoK98] Cristina Videira Lopes and Gregor Kiczales, Recent Developments in AspectJ™, Aspect-Oriented Programming (AOP) Workshop at ECOOP'98. 1998
[Pre95] Pree, W.; *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, Reading, MA, 1995
[Side00] SideFrame: Simple Design Pattern Framework, Homepage, http://www.cs.uni-essen.de/dawis/research/sideframe/
[Sou95] J. Soukup, *Implementing Patterns*. In: J.O. Couplin, D.C. Schmidt (eds.): Pattern Languages of Program Design, Addison-Wesley, 1995.

shanenbe@cs.uni-essen.de

Last modified: Aug 16 18:53:03 CET 2000