Redesign of Hot Spots using Aspect-Oriented Programming

Tapan Kant
Department of Computer Science
Banaras Hindu University
Varanasi-221005, India

Manjari Gupta
Department of Computer Science
Banaras Hindu University
Varanasi-221005,India

ABSTRACT
During last decade, software developers have given much more attention to the aspects and aspect-oriented programming (AOP). It offers a unique module to encapsulate scattered and tangled code. This approach might be helpful to solve the problem of crosscutting concerns. To the best of our knowledge, there are few reports are available in open source on design and programming part of framework, especially in modularising framework hot spots. Further, these reports are limited to the systematic approach for developing flexible hot spots. In this paper, we propose an aspect-oriented approach to redesign of hot spots by explaining how framework hot spots are related to the scattering and tangling problem. Further, we also introduced a comprehensive approach for de-scattering and un-tangling of hot spots using template-hook model in which these methods are rehabilitated into aspect. Our approach might be beneficial in the reusability of aspect-oriented implementation of a framework that is more flexible and modular. Besides that, the present study is suitable for being applied in available frameworks.

Keywords
Aspect-Oriented Programming, Software Reuse, Object-Oriented Framework, Hot Spots, Hook, Aspect-Oriented Framework.

1. INTRODUCTION
The redesigning of framework with aspect oriented paradigm becomes increasing common in comparison to object orientation. Given this trend, the new technology is based on flexible architectures that minimize the effort of developers to understand all the complexity inherent in a system.

A framework is a significant collection of collaborating classes that capture both the small-scale patterns and major mechanisms that implement common requirements and design in a specific application domain [1]. Framework allows designers and implementers in solving the problem at the design and code levels that can reduce the costs of developing applications. Earlier studies on object-oriented framework show the higher level of software reuse and implementation of techniques such as design patterns [2], meta-object protocols [3], and class reorganization [4] have supported in framework development.

Framework instantiation takes place at predefined point that is known as hot spots whereas fixed part is called frozen spots[5][6]. Hot spots comprise of template methods and hook methods [7]. Hook methods presents the variable part of the framework which is invoked by more complex method called template methods and represents stable part[2][6][8]. Typically, while adapting the framework one has to implement hook methods specific for the applications. Since, these set of methods are scattered around framework classes therefore it is difficult to identify set of template and hook methods to perform common task in different realm.

Nowadays, advancement in programming technology paradigm switch from classical object oriented programming (OOP) to AOP [9]. AOP is programming paradigms that preserves the ad-vantages of OOP and provide better separation of modules by avoiding tangled code and increases system maintainability. Therefore, the present study aimed to redesign hot spots using AOP. In order to support our work, we have presented a case study on an open source JHotDraw [10] framework with the implementing language AspectJ [11]. Overall, the present study is an attempt to solve the issue in hot spot at design level.

The rest of the paper is organized as follows. Section 2 describes back-ground, section 3 a brief introduction to AOP and AspectJ, section 4 describes our approach, we show our case study in section 5 and conclude our work in section 6.

2. BACKGROUND AND RELATED WORK
The object-oriented software development necessitates understanding about the combination of class inheritance and class composition to increase reusability at large scale. The techniques reported in [12][13][14] support the creation of reusable classes, and framework, also describe the dangers of class inheritance. Thereafter, the design pattern catalogues are introduced which provide some of the key building blocks for reusable object-oriented design [2][15][16].

Object-oriented frameworks [14] play a key role in solving the specific domain of problems and development of large-scale applications. Frameworks are collections of semi-complete classes that need to be customized for specific applications. The customization process of framework takes place through hot spots [2][6][8]. It can be adapted either through inheritance or by composition categorises as white-box and black-box, respectively. White-box framework consists of incomplete classes whereas meaningful implementation of classes is present in black-box framework [7].

Hot Spots are vital in any object-oriented design, it should be thoroughly understood while reusing for specific application. The basic design of hot spots is based on template methods and hook methods. Template methods are immutable and hook methods are mutable representing point of flexibility, mutually supports adherence to the Open-Closed Principle. The underlying principle states: an abstraction must be open for adaptation that often change and it must be closed to provide stability [17]. Template methods can be combined with hook methods by either inheritance or composition. Such a combination often resemble as Template Method design pattern [2][6][18]. Template method accentuates an invariant template method and links with the variant hook methods, whereas a Hot Spot focuses on variant hook method and associates the template methods from which it is invoked. The comparison between object-oriented and aspect-oriented implementation of the GoF patterns are showed aspect-orientation has advantage in properties like locality,

reusability, composition transparency [12]. Santos et al. [19] has proposed that framework hot spots can be modularized in terms of specialization aspects and these can give support for specializing a framework in step-wise manner to facilitate framework usage. JHotDraw [10] framework (implementation of Object oriented design patterns in Java) is a well-designed application model for aspect-oriented migration. AJHotDraw [20] (written using AspectJ and imply framework re-implementation) is an aspect-oriented refactoring of JHotDraw [10] which provides the feasibility of adopting aspect-oriented solutions.

3. ASPECT ORIENTED PROGRAMMING

According to Kiczales *et. al.* [21] AOP paradigm new evolution in the line of technology for separation of concerns that allows design and code to be structured to reflect the way developers want to think about the system. This approach was introduced by Kiczales *et. al.* in [9]. The principle behind this approach is separation of concerns. The basic elements of AOP are aspect, join points, advice and point cuts. Aspects are class like structure and can contain attributes, methods, etc; join point is a place in the program which can control the flow of a program. Advices are activated when a certain join point is reached. A set of join point which has to activate an advice is point cut.

AspectJ [10][11] is an extension of Java for AOP paradigm. It supports separate definition to concerns that affect system units. This separate unit allows better modularity by avoiding tangled code and increases system maintainability.

Aspect is the main construct of this language that crosscuts a system called crosscutting concern. An aspect can contain attributes, methods and may extend another aspect to implement concrete behaviour. They can affect the static structure of system by introducing fields or methods to an existing class and can change the hierarchical structure of classes.

Aspects have capability to change the dynamic object structure by intercepting certain points, called joint points. The flow of execution is modified on introducing additional behaviour before, after or around at joint points.

4. OUR APPROACH

The framework hot spots are based on template-hook model [7]. The template and hook methods belong to same class and adaptation can be made through inheritance is unification meta-patterns whereas the template and hook methods appear in different classes and adaptation takes place through composition is separation meta-patterns [23]. See figure 1 and 2. According to Reinhard et. al. [24], a set of hook methods and their associated template methods, in which each hook method is invoked by exactly the same set of template methods to perform some common task and categorizes hot spots into three different category as Inheritance Hot spots, Composition Hot spot and Hot spot.

Hook methods and template methods are scattered/tangled around different classes of framework. This scattering/tangling lead to difficulty in framework based application development. The application developer has to identify relevant hook methods specific to application. The scattering/tangling of template and hook methods results in modularity and variability problem. As shown in figure 1 and 2

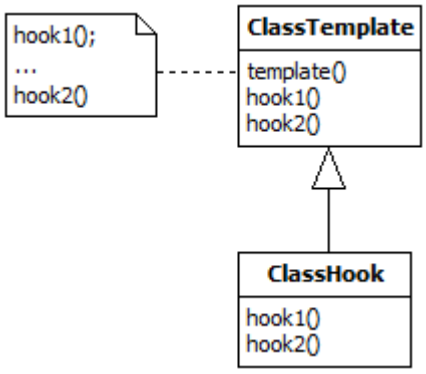

Fig 1: Unification Meta-pattern

In figure 1, the class ClassTemplate • contains template method and hook method (tangling problem). These hook methods are overridden in class ClassHook. In figure 2, the class Template contains template method whereas hook method is defined in separate class say AbstractHook (Scattering problem). The concrete classes ConcretHookA • and ConcreteHookB are subclasses of AbstractHook class.

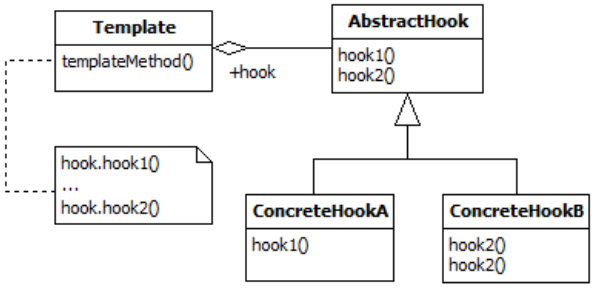

Fig 2: Separation Meta-pattern

In our approach, the classes that contain template and hook methods should be rehabilitated into aspect. Hanenberg et. Al. [25] have proposed Template Advice to specify template as advice. Similarly, we have also used to specify hook methods inside an advice instead of a method and a number of aspects are implemented for each join points. The following UML diagram (figure 3 & 4) show our approach.

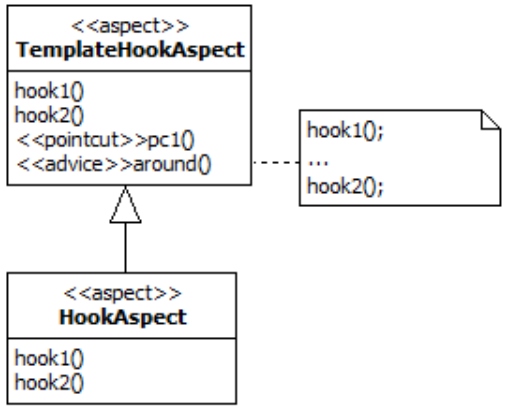

Fig 3: Solution to Unification Meta-pattern

Figure 3 depicts the solution to unification meta-pattern, the class "ClassTemplate" is transformed into aspect. The aspect contains only hook methods, a pointcut and an advice. The hook methods are called inside the advice.

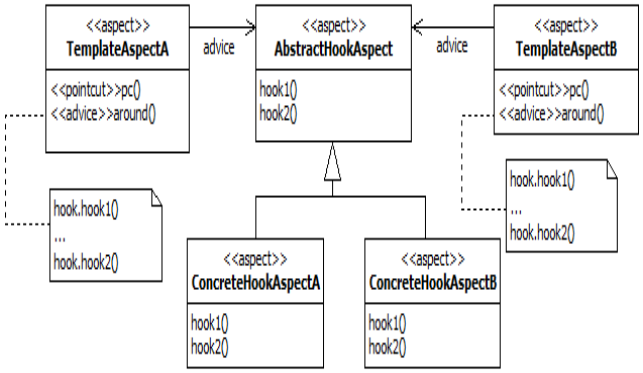

Fig 4: Solution to Separation Meta-pattern

In figure 4, two separate aspects have been developed as TemplateAspectA and TemplateAspectB for two different join points (i.e. ConcreteHookAspectA & ConcreteHookAspectB).

5. CASE STUDY: JHOTDRAW

JHotDraw [10] is a Java framework for applications that deal with technical and structured graphics. The framework can be considered of type gray-box, i.e. its extensibility is based both on white-box and black-box mechanisms. JHotDraw [10] allows applications to extend the core functionality with elements such as menus, menu items, figure types, action listeners, etc.

In our case study JHotDraw [10] framework has been used. We present a small application as an example to show the limitations of the traditional approach of framework hot spots in terms of modularization and extensibility.

Figure 5 show the class diagram for drawing a figure using JHotDraw [10] framework. The following diagram depicts that a drawing has figure, decorator figures and composite figures composed of connectors, handles etc. Any modification to a figure is reported to FigureListener class.

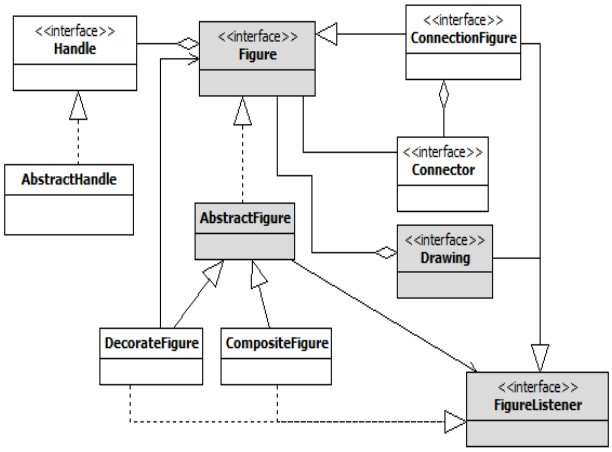

Fig 5: JHotDraw Framework (Figure class hierarchy)

The classes/interfaces Figure, AbstractFigure, Drawing & FigureListener represent hot spot as shown in figure 5 in grey colour. Here we can easily find separation meta-pattern. Figure class contains addFigureListener(), removeFigureListener(), changed(), etc. methods and FigureListener class contains figureChanged(), figureAttributeChanged(), etc. methods. The changed() method (template) of Figure class invokes figureChanged() method (hook) of FigureListener class whenever any change is made, so the relation between these two classes are similar to template and hook class as show in above figures (figure 1 & 2). The solution is obtained in figure 6.

In the above diagram (figure 6), the relationship aspect is used to maintain the hierarchical relationship between classes and interfaces. The benefit of this aspect is that an application developer can easily find the hierarchy on going through a single aspect. This aspect is only one per package.

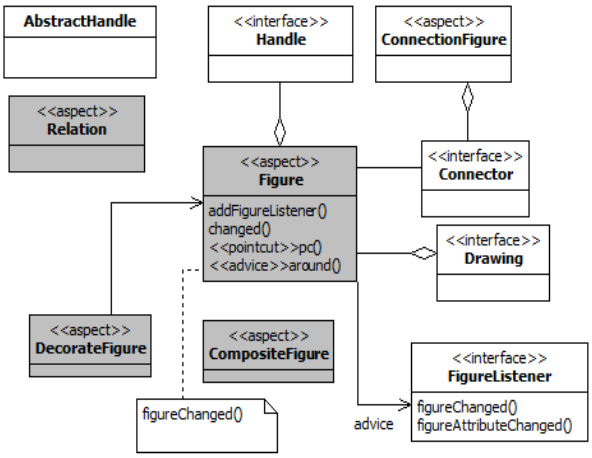

Fig 6: Redesign of Hot-spot (JHotDraw Framework)

The Figure class and their child classes are converted into aspect (i.e. DecorateFigure, CompositeFigure and ConnectionFigure). Since, we have identified template-hook model in figure 5 between Figure and FigureListener interfaces. A pointcut and an advice are introduced to Figure aspect. The hook method figureChanged() is called inside the advice.

In the above diagram, we have redesigned the hot spot to achieve more adaptability and modularity to framework. This would help application developers to implement hot spot in easiest way.

6. CONCLUSION

In this paper, we have presented the implementation of hook methods inside an advice could be better solution for framework implementation at the early stages of framework development. Aspect-oriented approach (able to modularize applications according to crosscutting concerns) was used for the redesign of framework hot spots. For this, we specify the hook methods inside an advice instead of a method and a number of aspects are implemented for each join points. In the presented case study, we have shown how this approach can work. However, more experience with this approach is necessary in order to evaluate the technique deeply, for instance, concerning its applicability, reusability and scalability. In summary, the present approach allows to develop hot spots with high feature cohesion, although there is still some substantial research and implementation of this in software development work ahead.

7. REFERENCES

[1] Michael Mattsson. Object-oriented frameworks - a survey of methodological issues. 1996.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. De-sign patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[3] GregorKiczales, Jim des Rivières, and Daniel G. Bobrow. The art of metaobject protocol. MIT Press, Cambridge, MA, USA, 1991.

[4] E. Casais. An incremental class reorganization approach. In ECCOP'92, volume 615, pages 114–132. Lecture Notes in Computer Science, 1992.

[5] Wolfgang Pree. Framework Patterns. SIGS Books, 1996.

[6] Wolfgang Pree. Design patterns for object-oriented software development. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[7] Wolfgang Pree. Hot-spot-driven framework development. In Summer School on Reusable Architectures in Object-Oriented software Development, pages 123–127. ACM, 1995.

[8] Rebecca J. Wirfs-Brock and Ralph E. Johnson. Surveying current research in object-oriented design. Commun. ACM, 33(9):104–124, 1990.

[9] GregorKiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In ECOOP - Object-Oriented Programming, volume 1241 of Lecture Notes in Computer Science, pages 220–242.Springer Berlin Heidelberg, 1997.

[10] Erich Gamma and Thomas Eggenschwiler, JHotDraw an Open Source GUI framework for technical and structured graphics, http://www.jhotdraw.org.

[11] AspectJ, Xerox Corporation, Copyright 1998-2001, Palo Alto Research Center, Incorporated, 2002-2003, http://eclipse.org/aspectj.

[12] Jan Hannemann and GregorKiczales. Design pattern implementation in java and AspectJ. In OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pages 161–173, New York, NY, USA, 2002. ACM.

[13] Ralph E. Johnson and William F. Opdyke. Refactoring and aggregation. In Proceedings of the First JSSST International Symposium on Object Technologies for Advanced Software, pages 264–278, London, UK, UK, 1993. Springer-Verlag.

[14] R. E. Johnson and B. Foote. Designing reusable classes. In Object-Oriented Programming, June/July 1988.

[15] Frank Buschmann, RegineMeunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-oriented Software Architecture: A System of Patterns. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[16] Dennis Kafura, Greg Lavender, and Doug Schmidt. Workshop on design patterns for concurrent, parallel, and distributed object-oriented systems. SIGPLAN OOPS Mess. 6(4):128–131, October 1995.

[17] Bertrand Meyer. Object-Oriented Software Construction. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.

[18] Allen Wirfs-Brock, John Vissades, Ward Cunningham, Ralph Johnson, and Lonnie Bollette. Designing reusable designs (panel session): experiences designing object-oriented frameworks. In OOPSLA/ECOOP '90: Proceedings of the European conference on Object-oriented programming addendum: systems, languages, and applications, pages 19–24, New York, NY, USA, 1991. ACM.

[19] André L. Santos, Antónia Lopes, and KaiKoskimies. Framework specialization aspects. In Proceedings of the 6th International Conference on Aspect-oriented Software Development, AOSD '07, pages 14–24, New York, NY, USA, 2007. ACM.

[20] MariusMarin, The Software Engineering Research Group (SERG), AJHotDraw an open source software, http://swerl.tudelft.nl/bin/view/AMR/AJHotDraw.

[21] TzillaElrad, Mehmet Aksit, GregorKiczales, Karl Lieberherr, and Harold Ossher. Discussing aspects of aop. Commun. ACM, 44(10):33–38, October 2001.

[22] GregorKiczales, Erik Hilsdale, Jim Hugunin, MikKersten, Jeffrey Palm, and William Griswold. Getting started with AspectJ. Commun. ACM, 44(10):59–65, 2001.

[23] Wolfgang Pree. Meta patterns a means for capturing the essentials of reusable object-oriented design. In Mario Tokoro and Remo Pareschi, editors, Object-Oriented Programming, volume 821 of Lecture Notes in Computer Science, pages 150–162. Springer Berlin Heidelberg, 1994.

[24] R. Schauer, S. Robitaille, F. Martel, and R.K. Keller. Hot spot recovery in object-oriented software with inheritance and composition template methods. In Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on, pages 220–229, 1999.

[25] Stefan Hanenberg, Rainer Unl, and Arno Schmidmeier. Aspectj idioms for aspectoriented software construction. In *Universittsverlag Konstanz*, pages 617–644, 2003.