# The Hot-Spot Relationship in OO Framework Design

*Marcus Felipe Fontoura, Edward H. Haeusler, and Carlos José Lucena*
Departamento de Informática
Pontifícia Universidade Católica
Rua Marquês de São Vicente, 225
22453-900, Rio de Janeiro, Brazil
[mafe, hermann, lucena]@inf.puc-rio.br

PUC-RioInf.MCC33/98 Setembro, 1998

**ABSTRACT**

The generalization from viewpoints is a key design concept in our framework design method. However, viewpoints are not generally designed to be flexible applications. The hot-spot relationship is a new relationship in object-oriented design that helps in this generalization, providing the flexibility properties in the derivation of the framework kernel. This paper presents our viewpoint-based framework design method, highlighting the role of the hot-spot relationship. Object calculus is used to formalize the relationship and prove that its semantics is provided by the design pattern essentials.

**Keywords**

Object-oriented framework design method, viewpoints, design patterns, flexibility properties, object calculus.

**RESUMO**

A generalização a partir de viewpoints é um conceito central no nosso método para design de frameworks. Contudo, viewpoints não são geralmente desenvolvidos para serem aplicações flexíveis. A relação hot-spot é uma nova relação orientada a objetos que ajuda nesse processo de generalização, fornecendo as propriedades de flexibilização para a derivação do núcleo do framework. Este artigo apresenta o nosso método de design de frameworks baseado em viewpoints, destacando o papel da relação hot-spot. Cálculo de objetos é usado na formalização da relação e para provar que a sua semântica é dada pelos design pattern essentials.

**Palavras-Chaves**

Método para design de frameworks, viewpoints, design patterns, propriedades de flexibilização, cálculo de objetos.

## 1 INTRODUCTION

Frameworks [7, 5] can reduce the costs of developing applications since it allows designers and implementers to reuse their previous experience on problem solving at the design and code levels. Prior research has shown that high levels of software reuse can be achieved through the use of object-oriented frameworks, since they capture the common aspects of a family of applications.

Although object-oriented software development has experienced the benefits of using frameworks, a thorough understanding of how to identify, design, implement, and change them to meet evolving requirements is still object of research [14]. Techniques such as design patterns [5, 10], meta-object protocols [9], refactoring [8], and class reorganization [2] have been proposed to support framework development and cope with some of the challenges.

This paper presents a design method for object-oriented software that combines frameworks and viewpoints [4], highlighting the use of the hot-spot relationship for providing the framework flexibility properties. The hot-spot relationship supports the integration of frameworks and design patterns, a new and challenging issue [13].

Object calculus [3] is used to formalize the relationship, showing that its semantics can be provided by the design patterns essentials or meta-patterns [10].

## 2 FRAMEWORK DESIGN METHOD

The approach to framework design presented in this paper is based on the idea that any framework design can be divided into two parts: the kernel sub-system design and the hot-spot sub-system design. The kernel sub-system design is common to all the applications that the framework may generate, and the hot-spot sub-system design describes the different characteristics of each application that can be supported by the framework. The hot-spot sub-system uses the method and the information provided by the kernel sub-system and may extend it.

Figure 1 illustrates the viewpoint-based design method through the identification of its artifacts and processes. The following sub-sections describe the elements involved.

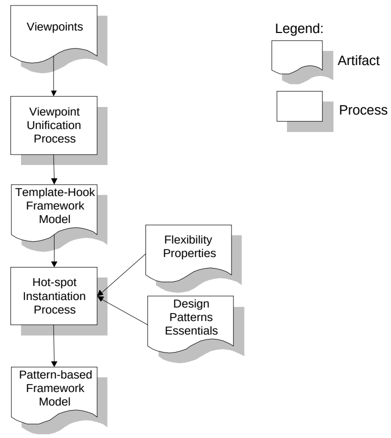


*Figure 1. Viewpoint-based design method process*

**Viewpoints**

The development of complex software systems involves many agents with different perspectives of the system they are trying to describe. These perspectives, or viewpoints, are usually partial or incomplete. Software engineers have recognized the need to identify and use this multiplicity of viewpoints when creating software systems [1, 6].

The first input artifact in the framework design method is a set of design diagrams that are developed based on perspectives, where a perspective defines a possible different use of the framework. In this way, a different system design associated with each different perspective is produced. In this paper, the term viewpoint is used to represent a standard object-oriented design associated with a framework perspective. The design is represented through the use of object-oriented diagrams, which can be developed using any OOADM (object-oriented analysis and methods) notation. This paper uses OMT [12] to illustrate the examples presented.

Figure 2 shows two viewpoints of a report generation application.

Viewpoint 1 main class is Report, which uses the classes GetData, to access the information required by the report, and GenerateReport, to generate the final report in an HTML file. The class GenerateReport uses the FormatReport class to configure the layout of the generated HTML file. Viewpoint 2 follows basically the same structure. The classes Report and GetData have the same methods, with same signature and implementation in both viewpoints. However, method generateReport (in class GenerateReport) has different signatures in the two viewpoints. One asks for an HTML file while the other asks for a RTF file.

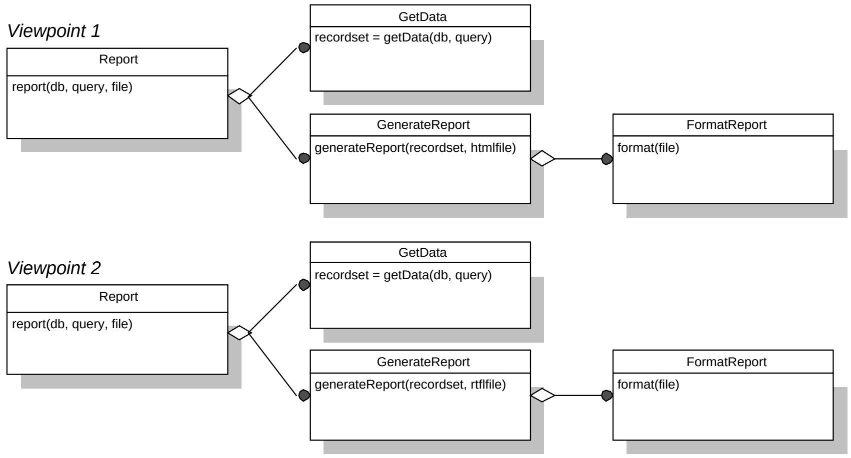


*Figure 2. Viewpoints of a same application*

**The viewpoint unification process**

In [11] Roberts and Johnson state that "Developing reusable frameworks cannot occur by simply sitting down and thinking about the problem domain. No one has the insight to come up with the proper abstractions." They propose the development of concrete examples in order to understand the domain. Our strategy is quite similar, analyzing each one of the viewpoints as a concrete example and deriving the final framework based on this analysis.

Once all the relevant viewpoints are defined the kernel design structure can be found by analyzing the viewpoints design representations and obtaining a resulting design representation that reflects a structure common to all viewpoints. This common structure is the "*unification*" of the viewpoints. This part of the design method is based on the domain-dependent semantic analysis of the viewpoints design diagrams to discover the common features of the classes and relationships present in the various viewpoints. The common part will compose the kernel design sub-system.

The elements that are not in the kernel are the ones that vary, and depend on the use of the framework. These elements define the framework hot-spots [10, 4] that must be adaptable to each generated application. Each hot-spot represents an aspect of the framework that may have a different implementation for each framework instantiation.

The unification process cannot automatically generate the final framework design since the viewpoints represent concrete applications, and for this reason the semantics of how to define the flexible part of the final framework is not present in their design. The design patterns essentials proposed by Pree [10] provide the appropriate semantics for the framework flexibility, since the software engineer can select the best design pattern constructor for each of the framework's hot-spots, considering their flexibility requirements.

A new relationship in object-oriented design, called hot-spot relationship, is used in the unification process to represent the relationships between kernel and hot-spot objects. The semantics of this relationship is given by the design patterns essentials. This implies that the hot-spot relationship is in fact a meta-relationship that is implemented through a design pattern that is generated taking into account the hot-spot flexibility requirements.

The design presented in Figure 3 is the result of the unification of the viewpoints presented in Figure 2. The dashed arrow represents the hot-spot relationship. The signature of the method generateReport is UNDEFINED, as established by the viewpoint unification rules [4]. This example shows viewpoints with similar design structures to simplify the method understanding, however complex unification rules and classes restructuring approaches [2, 8] can be used in a general case.

The UNDEFINED signatures and implementations are defined in the framework instantiation process through the use of domain specific languages (DLS) [4].

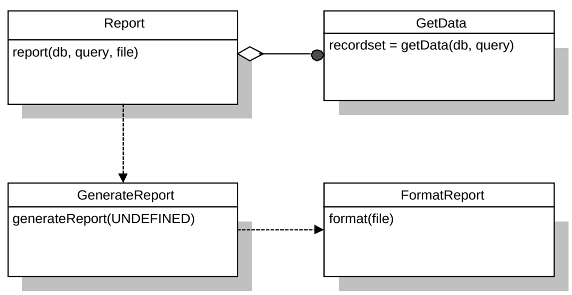


*Figure 3. Template-hook framework model*

**The hot-spot instantiation process**

This process uses the artifacts template-hook framework model, flexibility properties, and design patterns essentials as inputs and generates the pattern-based framework model as output. It is responsible for eliminating all the hot-spot relationships from the template-hook framework model, replacing then by the appropriate design pattern. The hot-spot cards guide the generation of an appropriate design pattern that will implement each hot-spot relationship, providing a systematic way for generating design patterns based on flexibility properties.

Figure 4 shows the hot-spot card layout, which is a variation to the one presented in [10]. The two flexibility properties used in the method are shown in the card: adaptation without restart and recursive combination.

Table 1 shows the mappings between the hot-spot card flexibility properties and the appropriate meta-pattern (or design pattern essential). In the unification meta-patterns, the template and hook methods belong to the same class and adaptations can be made only through inheritance, which requires the application restart for the changes to take place. In the separation meta-patterns, the template and hook methods appear in different classes and adaptations can be made at run time through the use of object composition in the same way of the strategy design pattern [5]. The meta-patterns design structures are presented through OMT diagrams in Figures 6, 7, 8(a), and 8(b), respectively.

| | Adaptation without restart | Recursive Combination | Meta-pattern |
|---|---|---|---|
| 1 | | | Unification |
| 2 | ☑ | | Separation |
| 3 | | ☑ | Recursive Unification |
| 4 | ☑ | ☑ | Recursive Separation |

*Table 1. Mappings between flexibility properties and meta-patterns*

The recursive combinations of template and hook methods allow the creation of direct object graphs, such as the composite design pattern [5].

As an example, let us consider the hot-spot relationships in the template-hook framework model shown in Figure 3. Suppose that it is necessary that new generateReport methods be defined by the system at run time. Also suppose that the format method does not need to be redefined at run-time. Since neither of these relationships require a recursive combination the hot-spot cards that represent their flexibility properties are presented in Figure 4. The resulting diagram, after the generation of the appropriate design pattern for each hot-spot relationship, is presented in Figure 5. This diagram is defined as the pattern-based framework model.

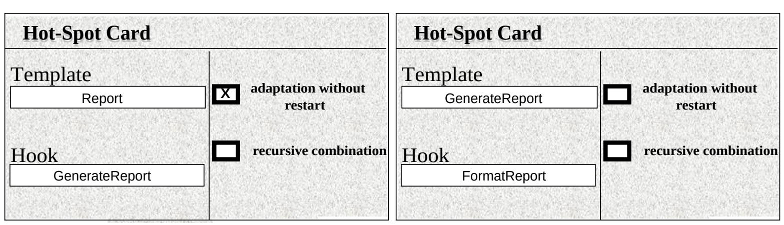


*Figure 4. Hot-spot card utilization example*

Since the design pattern generated for the hot-spot relationship between the classes GenerateReport and FormatReport unifies the template and hook methods in the same class and the FormatReport class only have the format method, this class is not needed in the pattern-based framework model. A very important property of the design method is the control of the design complexity, leading to a simple and readable design. In this example, the generated pattern-based framework model has less classes than each of the original viewpoints.

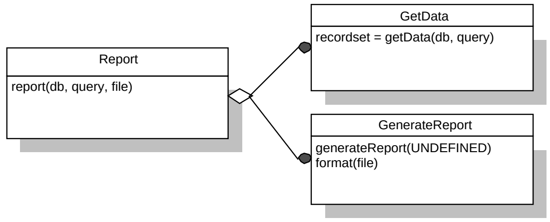


*Figure 5. Pattern-based framework model*

## 3 THE HOT-SPOT RELATIONSHIP

This section formalizes the hot-spot relationship properties through the use of object calculus [3]. For every pair of template-hook objects, a relationship is defined. The template object calls (or invokes) a method in the associated hook object. This constraint is stated by

$$\exists\, gt \in G_{t,},\ gh \in G_h \bullet gt \supset gh \qquad (1.1)$$

Where the template object is defined by the theory $T = \{S_t, A_t, G_t\}$ and the hook object is defined by the theory $H = \{S_h, A_h, G_h\}$. The following subsections specify the hot-spot relationship properties.

### Relationship Constraints

The hot-spot relationship establishes that various template objects can be associated with a hook object, while a hook object can be associated with various template objects. The only constraint for the existence of the relationship is that both objects have to be alive, as stated in axiom (1.2).

$$link(t, h) \Leftrightarrow t.create_T \wedge h.create_H \qquad (1.2)$$

### Flexibility Properties

As described in Section 2, two flexibility properties may be introduced by the hot-spot relationship: adaptation without restart and recursive combination. The combination of these two flexibility properties will lead to four design structure restrictions (Table 1).

The adaptation without restart means that the same template method can invoke more than one hook method during the system execution. If this property holds the lifetime of the template objects, hook objects, and hot-spot relationship cannot be the same, as stated in axiom (1.3). However, if the property does not hold their lifetime correspondence has to be the same.

$$t.kill_T \Rightarrow unlink(t, h) \wedge h.kill_H \Rightarrow unlink(t, h) \quad (1.3)$$

The recursive combination property establishes the property that the hook object can access the template objects methods. If this property holds we have axiom (1.4), which establishes a relationship from the hook object to the template object.

$$link(h, t) \qquad (1.4)$$

## 4 META-PATTERNS

This section describes the design structure of the meta-patterns presented in Table 1, and proves that the mapping showed in the table is sound.

### Unification

In the unification meta-pattern the template and hook methods belong to the same class, as showed in Figure 6. In this meta-pattern the hot-spot relationship is not transformed into any relationship, since the template object can access the hook object methods directly. Its formalization is shown bellow.

$$\exists\, gt, gh \in G_{th} \bullet gt \supset gh \qquad (u.1)$$

It is possible through this design structure that one template method be related to more than one hook method (and vice-versa), as stated by axiom 1.2. Since the methods belong to the same class their lifetime is the same and the adaptation without restart property does not hold. Axiom 1.4 also does not hold, since there is no relationship in this design structure. Thus, the first line presented in Table 1 is correct, once the unification meta-pattern preserves none of the flexibility properties.


*Figure 6. Unification meta-pattern*

**Separation**

The separation meta-pattern structure is showed in Figure 7. This pattern preserves axioms 1.1, 1.2, and 1.3. Axiom 1.4 does not hold, since there is no relationship in this design structure. Thus, the second line presented in Table 1 is also correct.

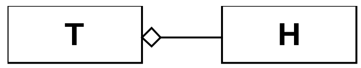


*Figure 7. Separation meta-pattern*

**Recursive Unification and Recursive Separation**

These meta-patterns design structure are presented in Figure 8(a) and (b), respectively. The soundness of Table 1 lines 3 and 4 is very similar to the ones previously shown, except that in these cases axiom 1.4 holds.

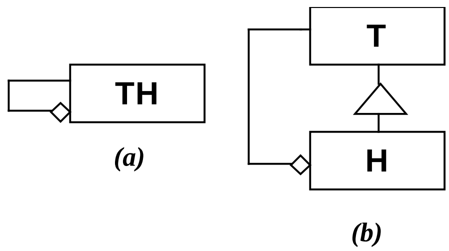


*Figure 8. Recursive meta-patterns*

## 5 RELATIONSHIP UNIQUENESS

This formalization shows that the hot-spot relationship cannot be directly implemented through any other existing object oriented relationship, such as aggregation, association, and inheritance, since in the three cases, axiom 1.4 does not hold. This fact shows that the hot-spot relationship is a meta-relationship that can only be implemented by a complete design structure, which are the meta-paterns or design pattern essentials.

## 6 CONCLUSIONS

This paper briefly presents our framework design method [4] and formalizes the hot-spot relationship, which plays an important role in the method: the introduction of flexibility properties in the process of generalizing framework abstractions from viewpoints. It proves that this new relationship can be implemented through meta-patterns, showing that each of the four meta-patterns preserves some desirable properties. The paper also shows that this new relationship is in fact a meta-relationship that cannot be directly implemented through the standard object-oriented relationships, like aggregation, association, and inheritance.

## REFERENCES


1. Ainsworth, M., Cruickshank, A. H., Groves, L. J., and Wallis, P. J. L., “Viewpoint specification and Z”, Information and Software Technology, 36(1), pp. 43-51, 1994.

2. Casais, E., “An incremental class reorganization approach”, ECOOP’92 Proceedings, volume 615 of Lecture Notes in Computer Science, pp. 114-132, 1992.

3. Fiadeiro, J. and Maibaum, T., “Describing, Structuring, and Implementing Objects, Lecture Notes in Computer Science, 489, Springer-Verlag, 1991.

4. Fontoura, M. F., Haeusler, E. H., and Lucena, C. J., “A Framework Design and Instantiation Method”, MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to ACM TOSEM) (This paper can be downloaded from the URL http://www.les.inf.puc-rio.br/~mafe.)

5. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. “Design Patterns, Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995.

6. ITU Recommendation X.901-904 – ISO/IEC 10746 1-4, “Open Distributed Processing – Referring Model Parts 1-4”, July 1995.

7. Johnson, R., “Frameworks = (Components + Patterns)”, Communications of the ACM, 40(10), 1997.

8. Johnson, R. and Opdyke, W. F., “Refactoring and aggregation”, Object Technologies for Advanced Software, First JSSST International Symposium, volume 742 of Lecture Notes in Computer Science, pp. 264-278, 1993.

9. Kiczales, G., Rivieres, J., and Bobrow, D., “The Art of Mataobject Protocol”, MIT Press, 1991.

10. W. Pree, “Framework Patterns”, Sigs Management Briefings, 1996.

11. Roberts, D. and Johnson, R. “Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks” in “Pattern Languages of Program Design 3”, Addison-Wesley, 1997.

12. Rumbaugh, J. Blaha, M., Premerlani, W., Eddy F., and Lorensen W., “Object-Oriented Modeling and Design”, Prentice Hall, Englewood Clifs, NJ, 1991.

13. Schmidt, D., Fayad, M., and Johnson, R., “Software Patterns”, Communications of the ACM, 39(10), October 1996.

14. Wirfs-Brock, R. and Johnson, R. “Surveying current research in object-oriented design”, Communications of the ACM, 33(9), 1990.