A Framework Development Method Using Viewpoints and the Views-a Relationship in the OO Design

Sergio Crespo, Marcus Felipe M. C. da Fontoura, and Carlos José P. de Lucena

Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brazil

e-mail: [crespo, mafe, lucena]@inf.puc-rio.br

PUC-RioInf.MCC13/98 Abril, 1998

ABSTRACT

In this paper we describe a viewpoint-based design method for framework development. This method uses the concept of viewpoints and the views-a relation in object-oriented design to guide the designer on the identification of hot-spots in the structure of the framework. Case studies have shown that high levels of software reuse can be achieved through the use of object-oriented frameworks. Although, analysis and design methods such as design patterns, meta-object protocol, refactoring, class reorganization, behavior of modification for framework, have been proposed to support frameworks development there are still problems related to this topic associated with ease of reuse. The viewpoint-based development method appears to be a very succesful approach from the point of view of component reuse.

KEY WORDS: object-oriented frameworks, viewpoints, framework development method, domain-oriented software development.

RESUMO

Neste artigo nós descrevemos um método para desenvolvimento de frameworks baseado em viewpoints. Este método usa o conceito de viewpoints e a relação views-a no design orientado a objetos para auxiliar o projetista na identificação de pontos de flexibilização na estrutura do framework. Estudos de caso têm mostrado que altos índices de reutilização podem ser alcançados com o uso de frameworks orientados a objeto. Apesar de métodos de análise e design como design patterns, meta-objetos, refactoring, reorganização de classes e modificação de comportamento terem sido propostos para auxiliar o desenvolvimento de frameworks, ainda existem problemas relacionados a esse tópico, principalmente em relação a reuso. O metódo de desenvolvimento baseado em viewpoints tem sido muito eficiente no aspecto de reutilização de componentes.

PALAVRAS CHAVES: frameworks orientados a objeto, viewpoints, metódos para desenvolvimento de frameworks, desenvolvimento orientado a domínio.

1. INTRODUCTION

Prior research has shown that high levels of software reuse can be achieved through the use of object-oriented frameworks [24]. An object-oriented framework captures the common aspects of a family of applications. It also captures the design experience in producing the application, and thus, allows designers and implementers to reuse this experience at the design and code levels.

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 are not well understood and still subject of research [17].

Techniques such as design patterns [10, 16], meta-object protocols [14], refactoring [13], and class reorganization [5] have been proposed to support framework development and cope with some of the challenges.

In this paper we formally present a method to object-oriented software development that combines frameworks and viewpoints [2, 3]. The method uses viewpoints and the views-a relationship [4] as key design concepts behind building the framework. The views-a relationship supports the integration of frameworks and design patterns, a still challenging issue [21].

2. A FRAMEWORK DEVELOPMENT METHOD

A framework [10, 12] is defined as generic software for an application domain. A Framework provides a reusable semi-finished software architecture that allows both single building blocks, and the design of the sub-system to be reused.

Our approach to framework development [2] is based on the idea that any framework design can be divided into two parts: the kernel sub-system design and the application-specific sub-system design. The kernel sub-system design is common to all the applications that the framework may generate, and the application-specific sub-system design describes the different characteristics of each application that can be supported by the framework. The application-specific sub-system uses the method and the information provided by the kernel sub-system and may extend it.

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

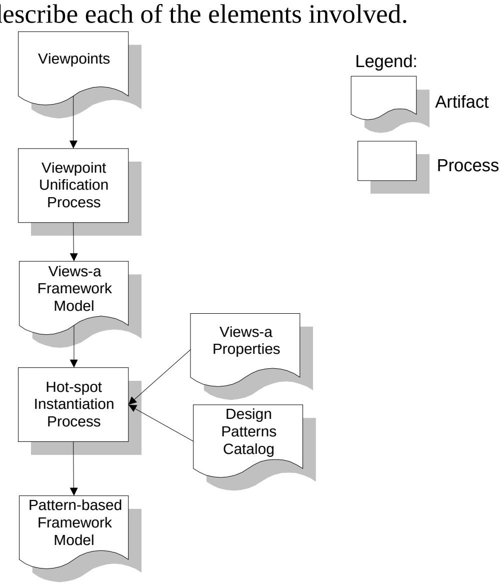

Figure 1. Viewpoint-based development method process

2.1 VIEWPOINTS

The development of complex software systems involves many agents with different perspectives of the system they are trying to describe. These views are usually partial or incomplete. Software engineers have recognized the need to identify and use this multiplicity of perspectives when creating software systems [1, 11].

The first input artifacts in the framework design approach are a set of viewpoints. We develop framework design diagrams based on perspectives, where a perspective defines a possible different use of the framework. In this way we will produce a different viewpoint associated which each perspective

(Figure 2). Viewpoints are represented as standard object-oriented class diagrams. These diagrams can be developed using any OOADM (object-oriented analysis and design methods) notation. We use the OMT notation [19] to illustrate the examples presented in this paper.

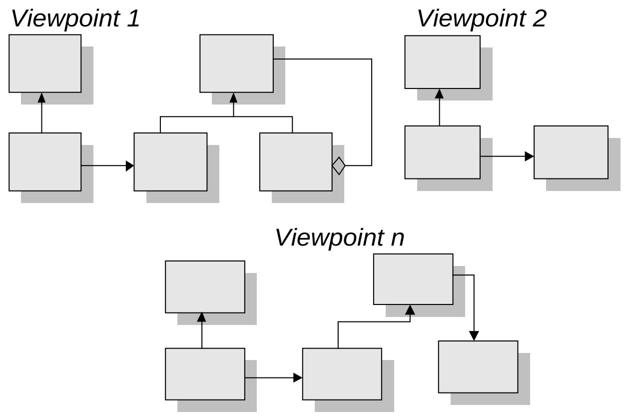

Figure 2. Different Viewpoints of the same Framework

2.2 THE VIEWPOINT UNIFICATION PROCESS

Once we have all the relevant viewpoints, we can discover the kernel structures by analyzing the design representations and obtaining a resulting design representation that reflects a structure common to all chosen viewpoints. We call this common structure the "*unification*" of the viewpoints. This part of the design approach is based on the domain-dependent semantic analysis of the design diagram to discover the common features of the classes and relationships present in the various viewpoints. The common part will compose the kernel 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 [16, 20] that must be adaptable to each generated application. The hot-spots will be used to define the structure of the application-specific sub-system. They can be identified through the same semantic analysis procedure based on the design diagrams that we used to identify the kernel. The identification of the kernel and its hot-spots is shown in Figure 3, in an abstract way.

Each hot-spot represents an aspect of the framework that may have a different implementation for each framework instantiation. A hot-spot can also be defined as a relation between an application-specific object and a kernel object in which the application-specific object uses the services of the kernel and must adapt them to its own view of the services. The method uses the views-a relation [4] to specify all the hot-spots in the system. By doing that a design diagram based on the views-a relationship is generated. This diagram is defined as views-a framework model.

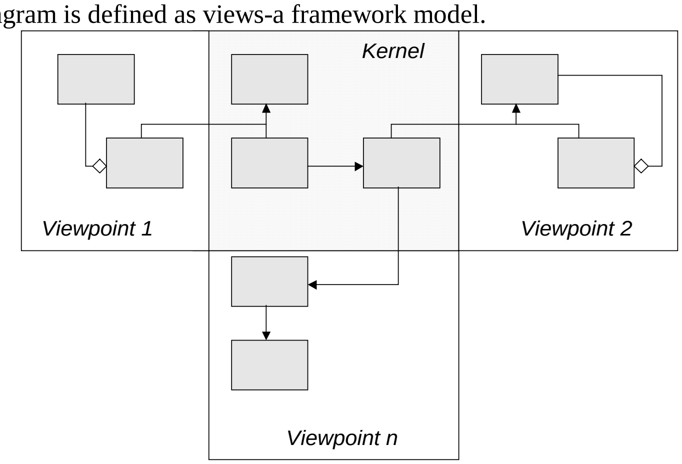

Figure 3. Unification of the Viewpoints

2.3 THE VIEWS-A FRAMEWORK MODEL

The views-a framework model is the result of the viewpoint unification process. This artifact is a representation, using views-a relation, of the kernel and its hot-spots.
The views-a relationship and its realization through the *Abstract Design Views (ADV)* [6] is a concept that promotes loose coupling between modules. It is maintained between two objects, where one object, called the view, monitor or "views" the states of another related object (viewed). The related object is unaware of the existence of the viewing object.
Each different kind of object-oriented relationship "glues" object in a particular way. The properties of this relationship characterize the static and dynamic constraints, which determine the type of interaction between objects, defining how one action occurring in one object influences other object.
In object-oriented approaches there are three basic kinds of relationship between classes: is-a, has-a and uses-a [18], and we extend this using the views-a relationship that models the interaction between two entities.
The views-a relationship is directed, and defines a set of constraints between the state and the behavior of the two connected objects. We characterize this relationship between views and viewed objects based on a set of primitive semantic properties that we now describe informally [4].

Properties	**Description**
Identity	View and viewed object should have different identities. This relation is irreflexive.
Viewed Singularity	Several views can monitor the same viewed object at the same time. However, a single view will monitor at most one viewed object.
Cardinality	The cardinality of a viewed class is exactly one, while the cardinality of the viewer class is in a range [n , m], where n and m are integers so that $0 \leq n \leq m \leq \infty$
Creation/Destruction	In a system, objects do not exist in isolation. Views is one relationship that creates a unidirectional dependency from one object to the other.
Attributes Consistency	We say that two objects are vertically consistent if their states are coherent in respect to the type of views relationship established between them. We say that viewer objects are horizontally consistent if each of them have attributes that are mapped to one or more common attributes in a same viewed object.

Table 1. Views-a Relationship Properties

We use the terms specialization to represent the is-a relationship, aggregation for the has-a relationship, and association for the uses-a relationship showing that the views-a relationship can not be directly modeled by these other abstractions. This comparison is based on the semantic properties of the views-a relationship.
These properties are the support to determine the behavior between objects using views-a relationship. We can demonstrate informally that the views-a relationship cannot be entirely simulated using specialization, aggregation or association. For example, the Identity property can be simulated by using aggregation or association, but cannot be simulated using specialization because an instance of a class is also an instance of the classes it specializes. In contrast, a view needs a different identity for its corresponding viewed object. The same can be done to all the other properties. A more detailed explanation of this informal proof can be found in [4].
The framework hot-spots are instantiated through views, which represent the application-specific objects. In this case, views can also be defined as components [12] that may be plugged into the hot-spots. The actual implementation of each views-a relation is made by the selection of a design pattern [10], in the hot-spot instantiation process.

2.4 THE HOT-SPOT INSTANTIATION PROCESS

This process uses the artifacts views-a framework model, views-a properties, and design patterns catalog as input and generates a pattern-based framework model as output. The selection of the best design pattern that implements each views-a relation is guided by the views-a cards [2], which provides a systematic way for selecting design patterns based on the views-a relationship properties described above. The mapping between each design pattern in the catalog and the set of views-a relationship properties it satisfies is presented in Section 4. More detailed examples can be found in [2].

Figure 4 shows the views-a card layout. Note that the Identity and Viewed Singularity properties are not present in the views-a card because they are not used in the design pattern selection since all patterns in the catalog satisfy them. Another important point is the introduction of the Recursive property, which is not a views-a property as described above but is a very important structural property from the implementation point of view. This property is very related to the 1:N recursive meta-pattern [16].

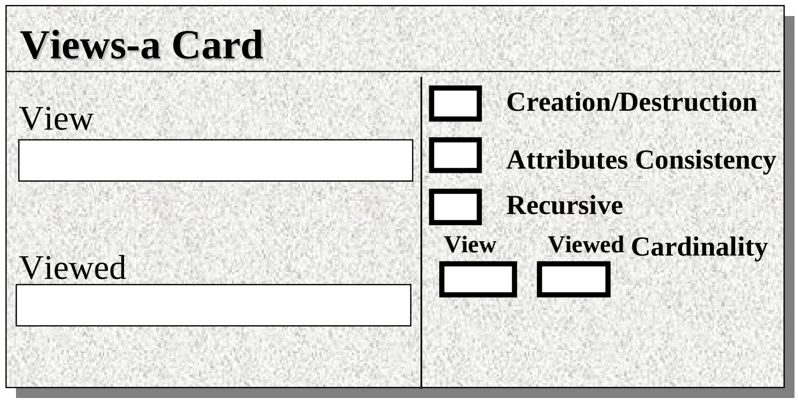

Figure 4. Views-a Card

2.5 THE PATTERN-BASED FRAMEWORK MODEL

The hot-spot instantiation process generates as a result a class diagram based on design patterns, which provide all the flexibility required for the framework's hot-spots. These patterns are representations of each views-a relation present in the views-a framework model. This resulting model can now be directly implemented through any programming language [10]. We are now working on transformation mechanisms for generating code in various platforms from the pattern-based framework model diagram [9].

3. AN EXAMPLE OF THE VIEWPOINTS UNIFICATION PROCESS

The viewpoints unification process step is responsible for generating a views-a framework model from a set of given viewpoints, which can be described through any OOADM notation.

This step imposes some pre-conditions on the viewpoints that are going to be unified:

- The viewpoints need to have name consistency, which means that classes with different names represent different concepts, and for this reason are not unified.
- The viewpoint's structure must be consistent. In this way, for example, two classes with same name and signature can not be related by different kinds of relationships in different viewpoints (Figure 5).

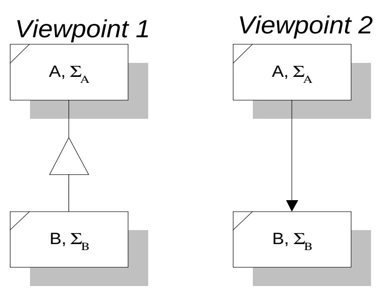

Figure 5. Viewpoint Inconsistency

In [7] the architecture and main features of an environment that supports the method is described. This environment has a viewpoint editor that guarantees the viewpoint consistency. In some cases some class restructuring approaches [5, 13] can handle the inconsistency.

When the set of viewpoints is consistent they can be unified. The unification process is based on the following rules:

1. If a class exists in more then one viewpoint with different signature it is defined as a hot-spot. Every hot-spot is defined as a view class.

2. If more then one class (different name) exist in more than one viewpoint with same signature, a refactoring [13] is performed to introduce a new abstract class, that is defined as a hot-spot (view class).
3. All the classes that are used by hot-spot class are defined as viewed classes. All the existing relationships between views and viewed classes are transformed into views-a relationships.
4. If one view class monitors more then one viewed class the viewed singularity property is broken. In this case, a refactoring is performed and a new design that guarantees that property is generated.

Figure 6 shows an example of three consistent viewpoints that are to be unified.

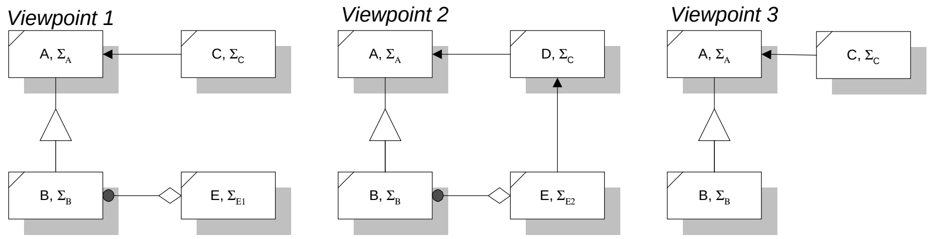

Figure 6. Viewpoint Unification Example

Class E is presented in viewpoints 1 and 2 with different signatures, Σ_{E1} and Σ_{E2} respectively. In this case E is defined as a hot-spot class, by the use of rule 1. We have now to look for the classes that E uses in all the viewpoints, mark these classes as viewed, and introduce a views-a relation between E, defined as view, and all the viewed classes (rule 3). In this example the viewed classes are B and D. The result of the application of rules 1 and 3 is presented in Figure 7, where the two-headed arrow represents the views-a relationship.

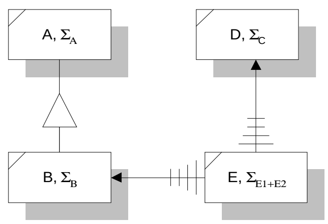

Figure 7. Same Name and Different Signature

Note in Figure 7 that the view class E monitors two viewed classes. In this case the viewed singularity property was broken and rule 4 must be applied. Its application leads to a refactoring step that introduces two new classes EB and ED, which will monitor classes B and D respectively. This new design is presented in Figure 8.

This refactoring process leads to a more modular design, since more classes are introduced to monitor specific viewed classes, which represent the kernel structure described in Section 2. Rule 4 preserves the *separation of concerns*, assuring that objects have specific and well-defined responsibilities [25]. Assuring this separation of concerns is one of the most important properties of our developing method.

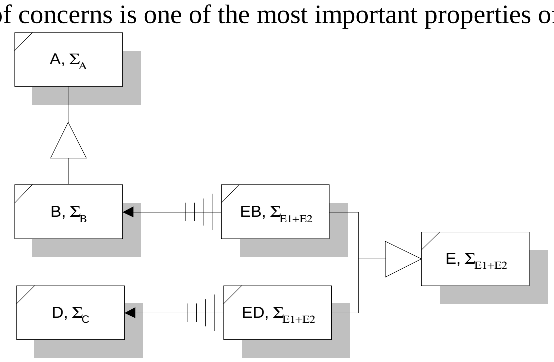

Figure 8. Viewed Singularity Restriction: Separation of Concerns

Now we have to look at the three input viewpoints and try to apply rule 2. We then find out that classes C and D have the same signature, Σ_C. In this case, a refactoring process is necessary. This process introduces the CD abstract class. Now applying rule 3 again, we mark as viewed classes all the classes that are used by either C or D, which in this case is just class A. The resulting diagram for this part of the process is presented in Figure 9.

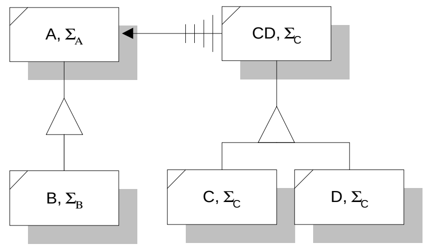

Figure 9. Same Signature and Different Name: Refactoring and Views-a

In this case rule 4 is not applied since the viewed singularity property has not been violated. The final product of the unification step, the views-a framework model, is composed of the diagrams presented in Figures 8 and 9.

4. AN EXAMPLE OF THE HOT-SPOT INSTANTIATION PROCESS

This process is responsible for eliminating all the views-a relationships from the views-a framework model, replacing then by the appropriate design pattern. The following table shows the mappings between the views-a card properties and the appropriate design patterns. The design patters used in the table are a variation of some patterns described in the GoF catalog [10]. The complete structure of these "new" patterns is presented in [2].

	Attributes Consistency	Cardinality	Recursive	Creation/ Destruction	Design Pattern
1		**1:1**			**State**
2		**1:1**		☑	**State**
3		**1:1**	☑		**Composite**
4		**1:1**	☑	☑	**Composite + State**
5		**1:N**			**Multiple State**
6		**1:N**		☑	**Multiple State**
7		**1:N**	☑		**Composite**
8		**1:N**	☑	☑	**Composite + multiple State**
9	☑				**Observer**
10	☑			☑	**Observer + State**
11	☑		☑		**Observer + Composite**
12	☑		☑	☑	**Observer + Composite + State**
13	☑	**1:N**			**Observer**
14	☑	**1:N**		☑	**Observer + Multiple State**
15	☑	**1:N**	☑		**Observer + Multiple State + Composite**
16	☑	**1:N**	☑		**Observer + Composite**

Table 2. Mappings between Properties and Design Patterns

As an example, we will consider the views-a relation between classes D and ED (Figure 8). Let us suppose[1] that this relation preserves the Attributes Consistency property but does not preserve the Creation/Destruction and Recursive properties. Let us also suppose that the cardinality is 1:N. Figure 10 illustrates the views-a card utilization for this example. In this case, the observer design pattern [10] is the more appropriate choice (Table 2 - line 13).

[1] In a real example the designer would have to select the desired properties for each views-a relation present in the views-a framework model. This step is supported by the views-a cards feature [2], and can be mechanicaly assisted [7].

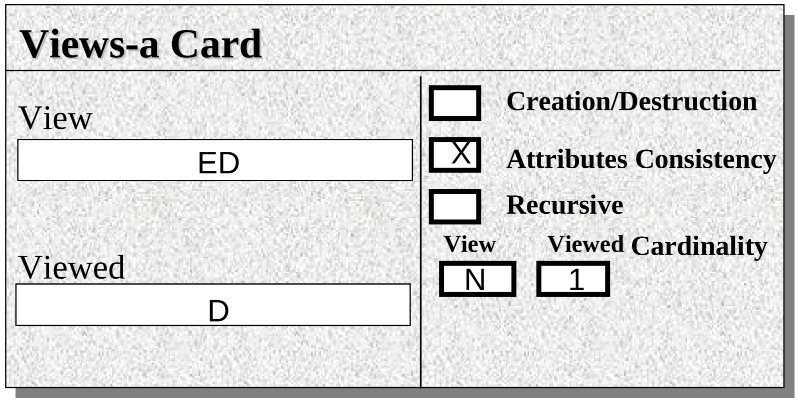

Figure 10. Views-a Card Utilization

Figure 11 shows the views-a relationship (a) and the OMT diagram for the observer pattern instantiated for this case (b). When all the views-a relations are removed, the design diagram generated is defined as a pattern-based framework model.

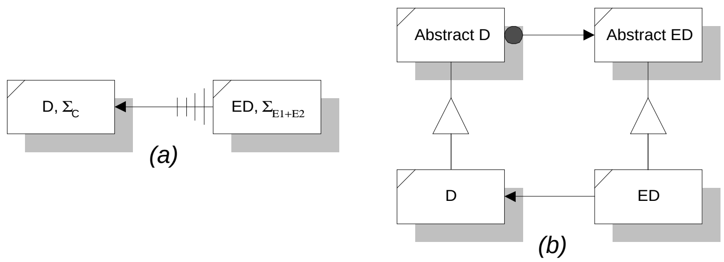

Figure 11. Hot-Spot Instantiation Example

5. CONCLUSIONS AND FUTURE WORK

This paper shows how viewpoints, and particularly, the views-a relationship can be used as the main design driving force to reusable framework construction. The method provides mechanisms that help the framework developer in the identification of the kernel structure and the flexible parts.

The views-a cards is presented here as an approach that can help us to bridge the gap between frameworks and design patterns, a still challenging issue [21].

The unification step is responsible for harmonically combining the various viewpoints. The use of the views-a relation in this step helps the definition of the framework kernel and its hot-spots. Other approaches to unification of viewpoints can be found in [8].

Another important characteristic of the method is that it is a complementary approach that can be combined to other techniques such as refactoring and OOADM. We are now developing a method support environment [7] that uses the refactoring algorithms presented in [13] for the viewpoint unification process and the OMT [19] notation for the viewpoint specification.

In [9] we present some approaches for generating code in various platforms based on the pattern-based framework model. We also present in [9] other examples of viewpoint unification and hot-spot instantiation.

We are also working on the method formalization [9], based on the Z specification language [22, 23]. Through this formalization we will be able to prove some important properties of the method, such as separation of concerns and loose coupling.

In [2] we present a case study for developing a framework for Web-based education. This framework is now fully implemented by the Software Engineering Laboratory, LES, of the Computer Science Department at PUC-Rio, Catholic University. An example of a application generated by this framework is the AulaNet™ environment (http://www.les.inf.puc-rio.br/aulanet) [15].

REFERENCES

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

2. P. Alencar, D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena, “A Viewpoint-based Design Method: A Case Study in Frameworks for Web-based Education”, MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998.

3. P. Alencar, D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena, “Using Viewpoints to Derive a Conceptual Model for Web-Based Education Environments”, MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to ASE’98).

4. P. Alencar, D. Cowan, C. J. Lucena, and L.C. Nova, “The Views Relationship in Object-Oriented Analysis and Design”, Technical Report, CS-97-13, University of Waterloo, Ontario, Canada, 1997.

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

6. D. Cowan and C. J. P. Lucena, “Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse”, IEEE Transactions on Software Engineering, March 1995.

7. S. Crespo, C. J. Lucena, and A. v. Staa, “A Meta-Environment for Framework Development”, MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998.

8. J. Derrick, H. Bowman, and M. Steen, “Viewpoints and Objects”, Technical Report, Computing Laboratory, University of Kent, Canterbury, UK, 1997.

9. M. F. Fontoura, E. H. Hermann, and C. J. Lucena, “Framework Development and Instantiation: a Formal Approach”, MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998.

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

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

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

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

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

15. C. Lucena, H. Fuks, R. Milidiu, L. Macedo, N. Santos, C. Laufer, M. Ribeiro, M. Fontoura, R. Noya, S. Crespo, V. Torres, L. Daflon, and L. Lukowiecki, “AulaNet™ - An Environment for the Development and Maintenance of Courses on the Web”, in ICEE’98, 1998 (to appear).

16. W. Pree, “Design Patterns for Object-Oriented Software Development”, Addison-Wesley, 1995.

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

18. James Rumbaugh, “Relational Database Design Using an Object-Oriented Methodology”, Communications of the ACM, 31(4): 417, April 1988.

19. James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, “Object-Oriented Modeling and Design”, Prentice Hall, Englewood Clifs, NJ, 1991.

20. H. A. Schmid, “Systematic Framework Design by Generalization”, Communications of the ACM, Volume 40, Number 10, October 1997.

21. Douglas C. Schmidt, Mohamed Fayad, and Ralph Johnson, “Software Patterns”, Communications of the ACM, 39(10), October 1996.

22. J. M. Spivey, “Understanding Z: A Specification Language and Formal Semantics”, Cambridge University Press, 1988.

23. J. M. Spivey, “The Z Notation: A Reference Manual”, Prentice Hall, Hemel Hempstead, 1989.

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

25. R. Wirfs-Brock, B. Wilkerson, and L. Wiener, “Designing Object-Oriented Software”, Englewood Cliffs, NJ, Prentice-Hall, 1990.