Viewpoints and Frameworks in Component-Based Software Design

P.S.C. Alencar, D.D. Cowan, T. Nelson
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
E-mail: [alencar,dcowan,torsten]@csg.uwaterlo.ca
M.F. Fontoura, C.J.P. Lucena
Department of Computer Science
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil
E-mail: [lucena,mafe]@inf.puc-rio.br

Separation of concerns is a well-established principle in software engineering that uses abstraction to hide complexity [13]. The importance of this principle increases dramatically as new technologies are introduced and as software applications and, in particular, frameworks, become more complex. Mastering the inherent complexity in the organization and design of object- oriented frameworks makes it necessary for developers to deal with an increasing number of 'concerns' which need to be separated [11].

Current frameworks involve a basic concern and a number of special purpose concerns. The basic concern is represented by the fundamental computational algorithms that provide the essential functionality relevant to an application domain, and the special purpose concerns relate to other software issues, such as user interface presentation, control, timing, synchronization, distribution and fault-tolerance. Special purpose concerns are extensions to the basic functionality that fulfill special requirements of the application, or enhance, manage, or optimize the basic algorithm [11]. Separation of these concerns localizes the different kinds of information in the software descriptions making them easier to write, understand, reuse, and modify.

The identification of the concerns relevant to a particular application and the definition of mechanisms to separate them has been an old challenge. As Dijkstra states: "The crucial choice is, of course, what aspects to study 'in isolation', how to disentangle the original amorphous knot of obligations, constraints and goals into a set of 'concerns' that admit a reasonably effective separation. ... " (E. Dijkstra, "A discipline of programming", 1976, last chapter).

Typical approaches to integrate concerns into the basic concern usually intertwine the fundamental algorithm with the special concerns. However, there are several problems that arise from this approach, including: first, there is very little design or code reuse because all concerns have to be dealt at the same time and at the same level; second, it is difficult to design, codify, understand, maintain and evolve systems in which all the concerns are strongly coupled.

To avoid strong coupling and promote design and code reuse, we separate the various concerns of the frameworks through "viewpoint" concepts throughout the design process. A viewpoint is a view of a system from a suitable perspective where we focus on a specific set of concerns. We could view a distribution system framework from a user or supplier perspective, or from the business rules that have been implemented. In contrast, we could also view a framework from a structural, functional or sequencing perspective. In the first case, we are examining the system from the functionality presented to the external world, while in the second case, we are examining the different relationships of the same aspect of the system.

The two types of viewpoints are called inter- and intra- viewpoint representations respectively. A key consideration of viewpoints is that multiple viewpoints must present the same system consistently. We choose to describe a viewpoint in terms of an object-oriented model, because of the rich constructs provided by this form of design language. The concepts associated with viewpoints are still evolving. For

example, the Unified Modeling Language (UML) [8] uses intra-viewpoints, while the Open Distributed Processing (ODP) Standard [14] uses inter-viewpoints.

In order to keep the concerns separated, we use a relationship called 'views-a' [3, 9]. Views-a can be used to model the connection between classes or components often called 'glue' in the literature [7, 12]. Using this abstraction to model glue allows the software developer to visualize the assembly operation more clearly. Normally, the glue would be buried in a complex programming structure that would obscure its real purpose.

The views-a operation has well-defined properties or semantics [3]. We characterize the relationship between viewer (also called ADV) and viewed (also called ADO) objects based on a set of primitive semantic properties. A rigorous definition of such properties, which can be seen as a formal model for gluing object-oriented components, is provided in [3, 6].

The views-a properties include the ones shown in the following table:

Views-a Property
1. Identity viewed and viewer have different identities
2. Cardinality the general cardinality is m viewers for one viewed
3. Creation/Deletion a viewed can outlive its viewers, a viewer can not outlive its views
4. Singularity/Multiplicity a viewed can have different viewers
5. Vertical Consistency views viewing the same state of a viewed must be consistent
6. Horizontal Consistency a viewed and a viewer object must be consistent
7. Visibility a viewed object does not know its viewers

The views-a operation can be implemented in terms of one of the standard object-oriented design patterns [10] such as the observer, mediator, or state. We choose the appropriate design pattern by comparing each design pattern with the views-a properties required in a specific design. For example, the observer or mediator pattern would be chosen to implement views-a if the viewed state of an object is changing and consistency among the viewed and viewers must be ensured over time. We have demonstrated the mapping of the views-a operation into different design patterns.

We propose a model for framework development based on viewpoints that includes the following five steps.

1. Determine the viewpoints of an application through viewpoint analysis where we can see each viewpoint as a subframework;
2. Determine the classes within a subframework and the subframeworks of an application that deal with the basic and the special concerns.
3. Glue both the classes within a subframework and the subframeworks together to create a complete application - the glue semantics is characterized by the views-a operation. Pre-built components can also be connected to frameworks by using this glue model.
4. Map the glue into design patterns - the required views-a semantics is used to guide the selection of appropriate design patterns. Object-oriented descriptions of the applications are produced.
5. Transform the resulting object-oriented design into "real" components - the interface and functionality of some components from an object-oriented perspective is described and the corresponding transformations are developed.

We have partially tested each of these steps by describing aspects of several applications. First, as a preliminary investigation, we have considered a case study that dealt with the NACHOS file system. In this case [5], there are two viewpoints: one for concurrency and one for large extensible files. Other case studies dealt with a process life-cycle application [2], a Web-based education software system based on viewpoints [1], and hypermaps [4].

We believe that our proposed approach to component-based design can be used even when the software developer uses ad-hoc methods. We have found through experimentation, that using some of the steps to analyze a design in progress clarifies many of the design decisions in the context of separation of concerns. Furthermore, an approach to framework development based on viewpoint concepts definitely improves the organization and leads to better designed resulting frameworks.

References

[1] P.S.C. Alencar, D.D. Cowan, S. Crespo, M.F. Fontoura and C.J.P. Lucena, "A Viewpoint-Based Design Method: A Case Study in Frameworks for Web-based Education", CS-98, Computer Science Department, University of Waterloo, 1998.

[2] P.S.C. Alencar, D.D. Cowan, M. Fontoura and C.J.P. Lucena, "A Framework Development Approach Based on Viewpoints", CS-97-33, Computer Science Department, University of Waterloo, 1997.

[3] P.S.C. Alencar and D.D. Cowan and C.J.P. Lucena, "A Logical Theory of Interfaces and Objects", revised for IEEE Transactions on Software Engineering, 1998.

[4] P.S.C. Alencar, D.D. Cowan, C.J.P. Lucena and M.A.V. Nelson, "An Approach to Hypermap-Based Applications", Proceedings of the Second International Symposium on Environmental Software Systems (ISESS'97), Whistler, British Columbia, pp. 244-151, 1997.

[5] P.S.C. Alencar, D.D. Cowan, C.J.P. Lucena and T. Nelson, "Viewpoints as an Evolutionary Approach to Software System Maintenance", CS-97-10, Computer Science Department, University of Waterloo, Proceedings of the International Conference on Software Maintenance, Bari, Italy, 1997.

[6] P.S.C. Alencar, D.D. Cowan, C.J.P. Lucena and L.C.M. Nova, "A Formal Theory for the Views Relationship", Proceedings of the 3rd Northern Formal Methods Workshop, Ilkley, UK, September 14-15, 1998.

[7] P.S.C. Alencar, D.D. Cowan and L.C.M. Nova, "A Model for Gluing Components", The Proceedings of the Workshop on Component-Oriented Programming (WCOP'98), (to appear), 1998.

[8] Rational Software Corporation, Rational, Unified Modeling Language version 1.1, 1998.

[9] Donald Cowan and Carlos Lucena, "Abstract Data Views: An Interface Specification Concept to Enhance Design", IEEE Transactions on Software Engineering, 21(3), March, 1995.

[10] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[11] Hursch, W. and Lopes, C. V., "Separation of Concerns", Technical Report NU-CCS-95-3, Computer Science Department, Northeastern University, Boston, MA, 1995.

[12] Gary T. Leavens, Oscar Nierstrasz and Murali Sitaraman, "1997 Workshop on Foundations of Component-based Systems", ACM SIGSOFT Software Engineering Notes, pp. 38-41, January, 1998.

[13] Roger S. Pressman", "Software Engineering—A Practitioner's Approach", New York, Montreal, McGraw-Hill, 1992.

[14] Open Distributed Processing, "Reference Model", http://www.iso.ch:8000/RM-ODP, ISO, 1998.

`