# Object-oriented application frameworks: the untold story

**Marcus F. Fontoura**
Software Engineering Laboratory (LES)
Computer Science Department, Pontifical Catholic University of Rio de Janeiro
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brazil
e-mail: mfontoura@acm.org

ABSTRACT

In the past few years software industry and academia have been devoting a lot of attention to object-oriented frameworks. Several frameworks have been proposed in the 90's and there are projections that application frameworks will be at the core of the technology of the twenty-first century. However, many development projects fail when trying to apply framework technology. This position paper describes some of the most common causes of failure and outlines possible solutions to overcome them.

## 1. THE MOST COMMON CAUSES OF FAILURE

The large number of framework projects that are being recently developed (e.g. ET++ [9, 17], Interviews [22], IBM San Francisco, Philips New York) show that organizations are trying to benefit from framework technology to achieve several economical advantages [10]. Theoretically frameworks may reduce development costs, provide faster time-to-market, and so on. However, real-world experiments in developing and applying framework technology are generally not so successful. There are several causes for the unsuccessful projects, varying from the intrinsic difficulty in finding the correct domain abstractions to model the framework architecture to the lack o appropriate supporting techniques and tools to support framework development and use. The rest of this paper addresses these issues.

### 1.1 FINDING THE CORRECT DOMAIN ABSTRACTIONS

An object-oriented (OO) framework [6] differs from a standard application because some of its parts, which are the hot-spots or flexible points [17], may have a different implementation for each framework instance, and are left incomplete until instantiation time. A framework models the behavior of a family of applications. Its kernel represents the similarities among the applications and the specific application behavior is provided by the hot-spots. However, it is very difficult to define the appropriate set of hot-spots for a given framework.

Roberts and Johnson propose the development of concrete applications before actually developing the framework itself [19]. They claim that framework abstractions can be derived from concrete applications. This may be a possible solution, but is very costly and perhaps unrealistic to be applied in real development scenarios. Research in the area of domain analysis [16] must still be developed in order to support the correct specification of framework hot-spots.

### 1.2 HANDLING FRAMEWORK COMPLEXITY

Frameworks can become quite complex and hard to understand. Especially when it is not clear what are the hot-spots, how they are related and how they should be instantiated. The IBM San Francisco, for example, provides approximately nine thousand classes and no description of what parts of its design are the hot-spots. It is estimated that the time for a regular developer to develop a real-world application based on San Francisco is six months, if he or she is already familiar with object-orientation and Java.

One possible solution for making the framework design more explicit and simple to understand is the development of more appropriate design languages. The approach proposed in [8] extends UML [15, 20] to explicitly represent the framework hot-spots and how the framework should be instantiated. Case studies have shown that this explicit representation enhances framework understanding, and helps the development process.

UML represents design patterns as collaborations (or mechanisms) and provides a way of instantiating framework descriptions through the binding stereotype [20]. Catalysis follows the UML approach and proposes a design method for building frameworks [4]. Both approaches also enhance framework design and instantiation.

## 1.3 SUPPORT FOR THE FRAMEWORK DEVELOPER

The implementation of framework hot-spots is one of the most critical parts in framework development. Several techniques such as design patterns [2, 9, 21], meta-level programming [13], and contracts [11, 12] have been proposed to overcome the lack of appropriate constructs to model flexibility and extensibility mechanisms in current OO programming languages. However, the selection of the most appropriate technique for each of the framework's hot-spots may be a very difficult task.

There are several UML case tools such as Together (http://www.oi.com), Rational Rose (http://www.rational.com) and Visio Enterprise (http://www.visio.com). These tools provide semantic checks that may be applied on the UML diagrams and provide nice visual interfaces for editing the design models by direct manipulation. However, they fail to support design concepts necessary to model frameworks, such as representing the hot-spots classes differently than the kernel ones.

Several design pattern tools [1, 5, 7, 14] have been proposed to facilitate the definition of design patterns, to allow the incorporation of patterns into specific projects, to instantiate design descriptions, and to generate code. However, they leave the selection of the most appropriate pattern to model each framework hot-spot in the hands of the framework designer. As shown in [8], tools that assist the systematization of the selection of the most appropriate modeling technique may be constructed, simplifying the job of the framework designer.

## 1.4 DELIVERING ADEQUATE DOCUMENTATION

Framework usability is a problem noticed in companies that use this technology to create applications, in part because of the intrinsic complexity of the instantiation process but also due to the lack of adequate documentation techniques and tools [3]. The most common way to instantiate a framework is to inherit from abstract classes defined in the framework hierarchy and write the code that is called by the framework itself. However, it is not always easy to identify, especially for non-expert users, which code is needed and where it should be written since class hierarchies can be very complex. Application developers have to rely on extra documentation to be able to create framework instances properly since it is very unlikely that they will be able to browse the framework's code and write the required instantiation code if no adequate documentation is provided.

Generally, framework instantiation is far more complex than simply plugging components into hot-spots. Hot-spots might have interdependencies, might be optional, frameworks may provide several ways of adding the same functionality, and so on. The instantiation process should be explicitly represented in an unambiguous form to allow application developers to create valid framework instances in a straightforward manner. This explicit process representation should also facilitate the construction of tools to assist framework instantiation.

### 1.5 SUPPORT FOR FRAMEWORK MAINTENANCE

Currently, there is very little tool support for framework maintenance and evolution. Some of the most common problems regarding framework evolution are described in [3] and summarized below:

- Structural complexity: framework evolution can make its structure (object interfaces, class hierarchies, and so on) hard to manage and understand;
- Changes in the domain: as the framework evolves and new framework instances are created, new abstractions that should be part of the framework may be derived;
- New design insights: framework design structure may need to be improved in the light of issues previously neglected or forgotten.

In all the three cases an explicit definition of the framework's hot-spots may facilitate the job of the framework's maintainer. Changing the implementation of an existing framework kernel class every time new functionality is needed, for example, is not good practice. If the hot-spots were appropriately marked, both framework and application maintainers would know how they relate to the rest of the system and which methods and classes can be removed without disturbing the framework or the instantiated applications.

Tools that support refactoring [18] and unification [8] transformations have been successfully used to support framework maintenance. Refactorings are behavior-preserving transformations that may be applied on the framework design representation as well as on the framework implementation. Unifications are transformations that alter the framework design structure to incorporate new features or modify the existing ones.

## 2. CONCLUSIONS

This position paper has presented some problems related to framework development and usability and briefly has described some possible solutions. Several of the problems presented here are cause of failure in framework projects developed in industry and academia.

The ultimate goal of my research is to define an adequate representation for frameworks that is useful in the documentation, implementation, and instantiation steps of the software development process. The proposed representation is complementary to existing OOADMs, and is defined an extension to UML. Detailed case studies of the use of the proposed notation to describe and implement real world frameworks and the specification, including the architecture and functionality, of an environment that supports the creation of frameworks using this notation is found in [8]. This environment supports design analysis over framework structure, generates code using various approaches, and generates documentation models. The first version of such an environment is completely developed and was used and validated in the development of several frameworks.

## REFERENCES

1. F. Budinsky, M. Finnie, J. Vlissides, and P. Yu, "Automatic Code Generation from Design Patterns", *Object Technology*, 35(2), 1996.

2. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, 1996.

3. W. Codenie, K. Hondt, P. Steyaert, and A. Vercammen, “From Custom Applications to Domain-Specific Frameworks”, *Communications of the ACM*, 40(10), 71-77, 1997.

4. D. D’Souza and A. Wills, *Objects, Components, and Frameworks with UML: The Catalysis Approach*, Addison Wesley, 1997.

5. A. Eden, J. Gil, and A. Yehudai, “Precise Specification and Automatic Application of Design Patterns”, ASE’97, *IEEE Press*, 1997.

6. M. Fayad and D. Schmidt, “Object-Oriented Application Frameworks”, *Communications of the ACM*, 40(10), 32-38, 1997.

7. G. Florijin, M. Meijers, P. van Winsen, “Tool Support for Object-Oriented Patterns”, ECOOP’97, *LNCS 1241*, Springer-Verlag, 472-495, 1997.

8. M. Fontoura, “A Systematic Approach for Framework Development”, Ph.D. Thesis, Computer Science Department, PUC-Rio, 1999.

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

10. D. Hamu and M. Fayad. “Achieving Bottom-Line Improvements with Enterprise Frameworks”, *Communications of ACM*, 41(8), 110-113, 1998.

11. R. Helm, I. Holland, and D. Gangopadhyay, “Contracts: Specifying Behavioral Composition in Object-Oriented Systems”, OOPSLA/ECOOP’98, Norman Meyrowitz (ed.), *ACM Press*, 169-180, 1990.

12. I. Holland, “The Design and Representation of Object-Oriented Components”, Ph.D. Thesis, Computer Science Department, Northeastern University, 1993.

13. G. Kiczales, J. des Rivieres, and D. Bobrow, *The Art of Meta-object Protocol*, MIT Press, 1991.

14. T. Meijler, S. Demeyer, and R. Engel, “Making Design Patterns Explicit in FACE – A Framework Adaptative Composition Environment”, ESEC’97, *LNCS 1301*, Springer-Verlag, 94-111, 1997.

15. OMG, “OMG Unified Modeling Language Specification V.1.2”, 1998 (http://www.rational.com/uml).

16. R. Prieto-Diaz and G. Arango (eds.), *Domain Analysis: Acquisition of Reusable Information for Software Construction*, IEEE Press, 1989.

17. W. Pree, *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995.

18. D. Roberts, J. Brant, and R. Johnson, "A Refactoring Tool for Smalltalk", University of Illinois at Urbana-Champaign, Department of Computer Science (http://st-www.cs.uiuc.edu/users/droberts/).

19. D. Roberts and R. Johnson, "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks", in *Pattern Languages of Program Design 3,* Addison-Wesley, R. Martin, D. Riehle, and F. Buschmann (eds.), 471-486, 1997.

20. J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1998.

21. J. Vlissides, *Pattern Hatching: Design Patterns Applied*, Software Patterns Series, Addison-Wesley, 1998.

22. J. Vlissides, "Generalized Graphical Object Editing", Ph.D. Thesis, Department of Electrical Engineering, Stanford University, 1990.