Pragmatic Issues Behind Framework Design

Paulo Alencar, Donald D. Cowan*, Sérgio Crespo♦, Marcus Felipe Fontoura♦, and Carlos José Lucena♦*

*Computer Systems Group
University of Waterloo
Waterloo, Ontario
Canada, N2L3G1
[alencar, dcowan]@csg.uwaterloo.ca

♦Departamento de Informática
Pontifícia Universidade Católica
Rua Marquês de São Vicente, 225
22453-900, Rio de Janeiro, Brazil
[crespo, mafe, lucena]@inf.puc-rio.br

ABSTRACT

This position paper describes our work in application frameworks. Some questions for discussion are also stated and we try to answer them based on either our theoretical studies or our previous experience. Our major research topics are also presented as relevant points to be discussed in the workshop.

KEY WORDS: object-oriented frameworks, framework design, framework instantiation, viewpoints, hot-spot, domain specific languages, formal methods.

1. INTRODUCTION

Prior research has shown that high levels of software reuse can be achieved through the use of object-oriented frameworks [31]. An object-oriented framework captures the common aspects of a family of applications. It also captures the design experience required while producing applications, and thus, it allows designers and implementers to reuse their 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 is still object of research [22]. Techniques such as design patterns [10, 21], meta-object protocols [15], refactoring [14], and class reorganization [4] have been proposed to support framework development and cope with some of the challenges.

We are currently working in a design and instantiation method for object-oriented software that combines frameworks and viewpoints. The method uses viewpoints and the hot-spot relationship as key design concepts for building the framework [2]. The hot-spot relationship supports the integration of frameworks and design patterns, a new and challenging issue [24].

The basic idea behind our framework instantiation method is to capture the domain concepts in a domain specific language (DSL) [11], which will help the framework user build the instantiation code in an easier without concerning for implementation decisions while remaining focused on the problem domain. The specification written in the DSL is then translated to the framework instantiation code through the use of transformational systems [17, 5].

The method is currently being formalized through the specification of each artifact and process involved. The specifications, presented as Z schemas [27, 28], are used to precisely describe each of the method's steps and to highlight its most important properties. We are also using object calculus [9] to formalize the framework flexibility properties and the hot-spot relationship.

A case study in the Web-based Education (WBE) domain was developed to validate the method. An object-oriented framework, called ALADIN, was developed from this case study. We have tried to capture the core functionality of various analyzed WBE environments [6, 16, 19, 25, 29, 30] to define an object-oriented framework. We have used the viewpoint unification [2] process to define the framework's kernel structure and it's hot-spots.

2. QUESTIONS FOR DISCUSSION

This section presents answers to the workshop questions and highlights some important aspects of our work.

2.1 When a framework provides an "approximate" fit to an application, how can a user expect to modify or extend the framework?

A framework [10, 13] is defined as generic software for an application domain. It provides a reusable semi-finished software architecture that allows both single building blocks, and the design of sub-systems to be reused.

Our approach to framework design 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.

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, 12].

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 (Figure 1). We use the term viewpoint 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.

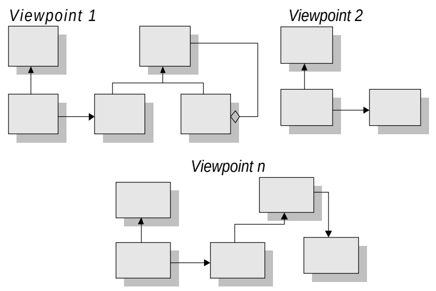

Figure 1. Different viewpoints of the same framework

In [22] 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 from 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 [21, 23] 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.

Thus, an object-oriented framework fits an application domain if the viewpoints used to derive its kernel structure reflect the relevant aspects of the domain, which compose the domain theory. To modify or extend the framework a user is expect to reanalyze what the relevant viewpoints are and redefine the framework kernel.

2.2 How can a framework support product evolution as the fundamental application domain changes?

The viewpoint unification analysis briefly described in Section 2.1 plays an important role in the domain redefinition. If a domain changes, new viewpoints will arise and the framework kernel will also change.

However, if the current version of the framework can cope with the new application requirements we propose the use of domain specific languages to generate the new application, which will be a new framework instance.

The most common way to instantiate a framework is to inherit from some 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 which code and where it should be written since frameworks class hierarchies can be very complex, especially for non-expert users.

Therefore, the "common" instantiation process is not always the ideal way to describe a particular application. This happens because of several facts:

- the language used to build and use the framework is a wide spectrum language, where it is not always easy to express the user intentions [26]. A short example would be the definition of event handlers in Microsoft MFC. The user has to write a macro to register her methods that will be called when a button is pressed or a list is scrolled. With an interface definition language [18], this task becomes easier and more intuitive;
- the complexity of framework class hierarchies and the difficulty of finding the points where the code should be written, that are the framework hot-spots or flexible points.

Our method proposes a different process for framework instantiation, where a particular application is described by domain specific languages (DSLs) [3, 8, 11, 20] that are designed precisely for the framework domain. The use of the DSLs allows the designer to develop quickly and effectively a complete software system [11].

The proposed instantiation process uses the following elements: DSLs, the framework, and a transformational system. The DSLs and the framework gather the domain main concepts. The transformational system is used to map the specification written in the DSLs to framework instantiation code. A customized application is the combination of the framework with its instantiation code. Figure 2 presents a flow diagram of this process.

As a precondition, this process requires the DSLs definition. The Draco-PUC [17] allows the definition of new languages by the specification of its grammar, in a BNF like notation, and a set of transformation rules to generate the framework instantiation code. The domain designer performs these steps. To generate a specific application a regular user needs to write a specification in DSL and then apply the set of transformation rules over it.

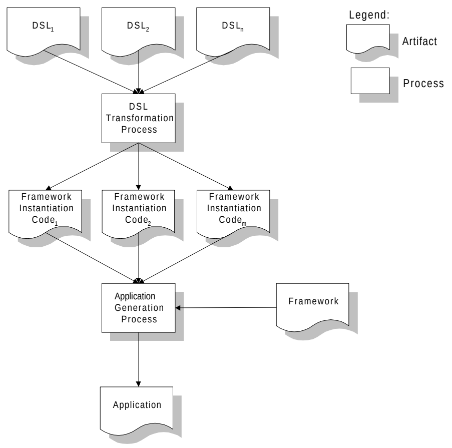
DSL1
DSL2
DSLn
Legend:
Artifact
Process
DSL Transformation Process
Framework Instantiation Code1
Framework Instantiation Code2
Framework Instantiation Codem
Application Generation Process
Framework
Application

Figure 2. Framework instantiation process

2.3 What types of problems arise wen multiple frameworks must be integrated? How do we go about structuring frameworks to ease the difficulty of integration?

The separation between kernel and hot-spot sub-systems, and the use of domain specific languages to instantiate the framework can facilitate the integration of frameworks.

With this approach the DSL encapsulate the framework structure and therefore the integration problems are limited to the definition of new languages that integrates the domain theories. That means that all the domains that are being integrated provide the semantics of these new DSLs.

2.4 In what application contexts is framework development desirable, or even practical?

There are various application areas that are not established yet and for which new ideas and models are presently under development and evaluation. These are domains in which the possibility of rapidly building new applications is essential and strategic from a practical point of view. Examples of application domains that can be classified in this category include Web-based Education, electronic commerce, biology, and financial market.

An interesting strategy for developing new applications for these domains is the development of object-oriented application frameworks. Therefore, framework development is very expensive not only because of the intrinsic difficulty related to capturing the domain theory but also because of the lack of appropriate methods and techniques to support framework specification and design.

2.5 What is the relationship between frameworks and COTS?

The black-box reuse of COTS systems can be interesting from the practical point-of-view. However, the reuse and extension of COTS systems depend very much on the system design and documentation and is very similar to the problem of integrating two frameworks.

If a COTS application is well structured and documented we can easily extend it or even define a new language, which will work as an API, to manipulate it.

3. CONCLUSIONS

In this position paper the concepts behind our viewpoint-based framework design method have been presented.

The paper also presents an approach to integrate frameworks with domain specific languages (DSL). We argue that DSLs allows the domain expert to formalize the specification of a software solution immediately without worrying about implementation decisions and the framework complexity. To implement this approach a transformational system (Draco-PUC) is used to transform DSLs specifications into framework instantiation code. It is important to note that DSLs could be transformed into other DSLs, thus creating a domain network, in a way similar with described in [20], providing an easy implementation path for new DSLs.

The method seems to be useful in situations where the frameworks are very complex to build and difficult to use. It will be tested in important and innovative domains, in which we also expect to contribute with the design and development of real frameworks. Two domains that we plan to study soon are electronic commerce and computational biology.

We are now working in the derivation of domain specific languages from problem theory, which seems to be an interesting approach not yet been exploited in literature.

Since the Web-based Education (WBE) domain is still not completely understood the need for a framework that supports fast development of alternative WBE environments by non-programmers is a desirable goal. The ALADIN framework [7] developed as a case study for the method seems to be an environment in which teachers and education researchers can develop their own environments, with little help from software engineers.

A new version of the AulaNet™ environment (http://www.les.inf.puc-rio.br/aulanet) [19] is now being developed with the ALADIN framework. This experiment has two main purposes: the development of a more flexible version of AulaNet™ and further validation of the ALADIN framework.

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, "Using Viewpoints to Derive a Conceptual Model for Web-Based Education Environments", MCC17/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to Journal of Systems and Software).
3. J. Bell et al, "Software design for reliability and reuse: A proof-of-concept demonstration", In TRI-Ada '94 Proceedings, pages 396-404, ACM, Nov. 1994.
4. E. Casais, "An incremental class reorganization approach", In ECOOP'92 Proceedings, volume 615 of Lecture Notes in Computer Science, pages 114-132, 1992.
5. J. Cordy and I. Carmichael, "The TXL Programming Language Syntax and Informal Semantics", Technical Report, Queen's University at Kinkston, Canada, 1993.
6. D. Cowan, "An Object-Oriented Framework for LiveBOOKs", Technical Report, CS-98, University of Waterloo, Ontario, Canada, 1998.
7. S. Crespo, M. F. Fontoura, C. J. Lucena, and L. M. Moura, "ALADIN: An Architecture for Learningware Applications Design and Instantiation", MCC34/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to WWW Journal).
8. E. W. Dijkstra, "The humble programmer", Communications of the ACM, 15(10), Oct. 1972.
9. J. Fiadeiro and T. Maibaum, "Sometimes 'Tomorrow' is 'Sometime'", Temporal Logic, volume 827 of Lecture Notes in Artificial Intelligence, pages 48-66, Springer-Verlag, 1994.
10. E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.
11. P. Hudak, "Building Domain-Specific Embedded Languages", Computing Surveys, 28A(4), ACM, 1996.
12. ITU Recommendation X.901-904 – ISO/IEC 10746 1-4, "Open Distributed Processing – Referring Model Parts 1-4", July 1995.
13. R. Johnson, "Frameworks = (Components + Patterns)", Communications of the ACM, Volume 40, Number 10, October 1997.
14. 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.
15. G. Kiczales, J. des Rivieres, and D. G. Bobrow, "The Art of Mataobject Protocol", MIT Press, 1991.
16. LearningSpace, <http://www.lotus.com/home.nsf/welcome/learnspace>.
17. J. C. S. P. Leite, M. Sant'anna, and F. G. Freitas, "Draco-PUC: a Technology Assembly for Domain Oriented Software Development"; Proceedings of the 3rd IEEE International Conference of Software Reuse, 1994.
18. C. Levy, D. Cowan, C. J. Lucena, M. Gattass, and L. H. Figueiredo, "IUP/LED: A Portable User Interface Tool", Software Practice and Experience (accepted for publication).
19. 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^{TM} - An Environment for the Development and Maintenance of Courses on the Web", International Conference on Engineering Education, Rio de Janeiro, Brazil, 1998.
20. J. M. Neighbors, "The Draco Approach to Constructing Software from Reusable Components", IEEE Transactions on Software Engineering, vol. 10, no.5, Sept. 1984.
21. W. Pree, "Design Patterns for Object-Oriented Software Development", Addison-Wesley, 1995.

22. 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.

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

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

25. Simon Fraser University, <http://virtual-u.cs.sfu.ca/vuweb/>.

26. C. Simonyi, “The Death Of Computer Languages, The Birth of Intentional Programming”, Technical Report MSR-TR-95-52, September 1995.

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

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

29. University of British Columbia, <http://homebrew.cs.ubc.ca/webct/>.

30. Virginia Commonwealth University, <http://views.vcu.edu/wcb/intro/wcbintro.hml>

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