Choosing an Object-Oriented Domain Framework

Garry Froehlich*, H. James Hoover, Ling Liu, Paul G. Sorenson
Department of Computing Science
University of Alberta
Edmonton, AB. T6G 2H1
{garry,hoover,lingliu,sorenson}@cs.ualberta.ca

Version 1.5

1998 April 9

Abstract

Deciding whether or not a framework 'fits' an application and provides an appropriate basis for development of the application is one of the key decisions developers make when choosing to use a new framework. In this paper we outline a three step process for helping to determine whether or not a framework is appropriate. The process looks at the limitations, the hooks and the amount of uncertainty in using a framework.

1 Introduction

Object-oriented frameworks [2] enable developers to rapidly produce new applications — provided that the framework is actually suited to the requirements of the new application. Often, previous experience with the framework is used to make this decision, but when application developers are unfamiliar with a new framework they have no experience for deciding whether or not to use it. They may not discover that a framework lacks support for key requirements of an application until well into the development cycle, resulting in substantial redevelopment or discarding of the project altogether. They require a basis for making the initial decision of whether or not to invest time in understanding and using the framework. Our approach is most successful for domain specific object-oriented frameworks. Due to their size and overall generality, this approach may not be as applicable to enterprise frameworks.

*Main contact author, garry@cs.ualberta.ca

We have been working with existing frameworks such as HotDraw [6], as well as developing new frameworks in the engineering domain (EAF). Our experience, common to most framework users, is that building up the level of expertise necessary to know in detail what applications a framework can be used for is a time consuming process due to the abstract nature and complexity of most frameworks. How can this expertise be captured and be made available to new users of the framework to help them decide whether or not to use the framework? In other words, how do we describe the *applicable domain* of the framework?

A new application can fall into one of three areas. It can be (1) clearly outside of the applicable domain and thus the framework can be immediately rejected as being unsuitable; (2) clearly within the applicable domain and thus ideally suited to the framework; or (3) somewhere in between in a region of uncertainty in which using the framework entails a high degree of risk. When what can and cannot be done with the framework is poorly defined, the region of uncertainty is very large, and the decision of whether or not to use the framework is much more difficult.

The region of uncertainty can be reduced by better defining the applicable domain, but it can never be eliminated in a non-trivial framework. When an application falls into the region of uncertainty, the potential user of a framework will often perform experiments to increase their understanding of the framework and to evaluate its appropriateness to the new application. The framework should provide support for this experimentation.

2 Decision Process

There are three general steps for deciding whether or not to use a framework for an application.

1. Determine if the framework should be immediately rejected. Verify that the stated limitations of the framework do not violate any of the requirements of the application.

2. Determine if the framework is clearly suitable. Try to map the requirements of the application to the hooks described for the framework. The hooks [3] show what the framework is intended to be used for and describe how a framework can be extended to meet a given requirement. If there is a hook for every requirement of the application, then the framework can be easily used to develop the application.

3. Assess the level of uncertainty. If an application has requirements for which there are no hooks defined, then it falls into the region of uncertainty. The framework may be suitable for the application, but is not guaranteed to be so. Further, the greater the amount of uncertainty, the more likely it becomes that developers will need to put more work into implementing their application from the framework.

Support for each of the three steps is described below.

2.1 Limitations - What cannot be done with the framework.

Every framework will have limitations placed on it through design decisions made by the framework developers. The limitations of a framework generally cannot be easily changed or bypassed and can cause a project to fail if the application developers are not aware of them. Some applications will clearly fall outside of the stated domain of the framework, for example applying our engineering framework to payroll applications. Others are not so clear and the limitations of the framework need to be examined. Unfortunately, frameworks frequently have little or no documentation on their limitations, and determining them often requires inspection, experimentation, or appeal to an expert user.

Design patterns [4] often discuss the limitations and tradeoffs that are made when using the pattern and when design patterns are used to document design decisions in the framework [1]. The limitations of the design patterns then apply to the framework as well.

As an example, we'll use the SizeMaster application for selecting the appropriate pressure relief valve for a given situation. SizeMaster was built using our engineering application framework (EAF), and our business application framework (Kalos). Briefly, some of SizeMaster's requirements are to perform moderately complex sizing calculations, to show who performed and approved a given calculation, to maintain the units used in a calculation in an internally consistent way, to enforce the workflow of an order from requirements capture to selection of the appropriate valve, and to maintain a customer, order and valve database.

The engineering application framework is designed for a particular style of calculation found in many engineering standards, in which a relatively small number of variables and formulas can be inter-related in many ways. The framework imposes some structural limitations on the calculation (for example, it requires acyclic calculations in which two or more variables cannot each depend on the other), and has performance limits to the number of equations and variables used. The framework is certainly not intended for real time calculations.

Upon inspection, the valve equations from the design standard were found to fit within the limitations imposed by the EAF. Not all equation dependencies were acyclic, but the cases in the standard where this occurred were easily rectified by introducing some redundancy. The number of equations were also found to fall under the performance limits of the framework, although they indicated that object creation in the EAF is very expensive.

The associated user interface and persistent object framework provided by Kalos connects a database to a forms-style user interface. Kalos requires a relational database that supports SQL queries, and the forms transactions use optimistic locking. None of the requirements of SizeMaster clearly contradict these limitations as the application will use a database, can use a forms based interface and the model is such that conflicts will occur infrequently so optimistic locking is appropriate. The frameworks cannot be immediately

rejected.[1]

2.2 Hooks - What can be done with the framework.

Techniques such as cookbooks [7], patterns [6], and hooks have been developed to help describe how the framework can be used. The hooks approach is the most structured and defined of the three and provides a good basis upon which to make decisions about framework adoption. Hooks occur at the hot spots [8] of a framework. Each hook gives a specific requirement that it fulfills and then documents how to extend the framework to meet the requirement. Hooks fall into three categories of use.

- Option: components are provided with the framework that can be easily used together within the overall framework architecture. For example, the Kalos framework provides a standard set of navigation buttons and menu options which can be used in an application.
- Pattern: components are configured with parameters and/or connected together in well defined ways. For SizeMaster, one of the appropriate patterns are the standard forms in the frameworks which are parameterized to display and allow the modification of customer and order records. Another pattern that fits the requirements of the application is the new worksheet hook which allows arbitrary equations and their dependencies between one another to be used in the application. There are also collaboration patterns that capture the intended ways of combining the EAF and Kalos frameworks.
- Open: the framework is extended in well defined ways, generally by providing new components. Within Kalos, open hooks are defined for creating the appropriate database tables, and persistent business objects with application specific behavior. In the case of SizeMaster, the business objects identified above are customers, orders and valves.

By matching the requirements of the application to the hooks provided with the framework, the application developers can be certain that their application will not "break" the framework. Although each hook represents an intended use of the framework, frameworks are also meant to be flexible and it is not possible to describe every conceivable use of the framework.

When using the decision process, developers should keep in mind that they give up a lot of design control to the framework. The requirements of the application have to be mapped to the capabilities and hooks of the framework in order to use the framework. The framework may fulfill the requirements in a different, and possibly better, way than the developers are accustomed to.

Some requirements of SizeMaster have not been accounted for by the hooks. Some of these are default features of the framework, such as requiring users to log in and tracking which user performed what action,

[1] Of course, both the EAF and Kalos were designed for just this style of application.

or keeping all units in a calculation internally consistent. Others, such as workflow fall in the region of uncertainty. For example, many applications have common patterns of workflow captured in wizards. Whether a wizard can exercise the required control over the workflow style supported by the framework is a significant area of uncertainty.

2.3 Dealing with uncertainty.

When appropriate hooks are missing for some of the requirements of the application, it is not clear if the framework is applicable. Often, application developers simply don't have enough knowledge about the framework to decide if it is suitable. They will use the framework in ways the original framework developers never thought of. They may decide that modifications to the framework are required when in fact the desired capability may already exist in the framework but was never documented. In our frameworks an application's workflow must fit into the general workflow model supported by the framework.

Since it isn't possible to completely remove the area of uncertainty around the applicable domain of a framework, developers need help to deal with the uncertainty. Within the region of uncertainly, application developers must have a deeper understanding of the architecture and implementation of the framework. A domain expert can help determine if the application is within the general domain of the framework, but may not know if it is within the framework's applicable domain. Clear documentation of the design of the framework, such as through design patterns or architecture description languages, can help to provide the understanding needed. In our experience, access to the source code of the framework is indispensable especially if new capabilities need to be added to it. In the case of SizeMaster, investigating the workflow requirements showed that some evolution of the framework was required.

Tools for understanding how the framework operates, such as exploring exemplars [5], can help deepen the developer's knowledge of the framework. Tools which aid in the rapid development of applications, such as the interactive development environments (IDEs) associated with graphical user interface frameworks, allow developers to quickly prototype their applications and find potential pitfalls. Implemented on top of Borland's Delphi framework, EAF and Kalos makes use of the Delphi IDE to enable developers to quickly prototype the basic user interface and the support for workflow required. The experience that comes out of these experiments can then be captured as new hooks or limitations of the framework, or simply as a description of what did or did not work. To guide the experimentation and learning process, we provide a Kalos standard application. It illustrates all of the key hooks in the framework without many of the obscuring details associated with a full application.

When significant use of open hooks is required or actual code changes to the framework are necessary, then the risk of framework use is high. Further exploration is necessary into tools to provide on-going advice and risk assessment for the framework user.

2.4 Other Factors

The process outlined only helps to determine the fit of the framework to the application requirements. There are other factors, beyond the scope of this paper, which also impact the decision of whether or not to use a particular framework. For example, a framework which has been used extensively will be more mature and stable than a newly developed framework and will likely incorporate good abstractions. Performance issues can also play a role. Tools provided with the framework can make it significantly easier to use than one with no such support. User support from the vendor can also be important. Finally, the understandability of the framework is critical to its use.

When choosing a framework which will be used for several applications, or across an entire business, these other factors are more critical than when choosing a framework for a single application. For example, vendor support becomes much more crucial when a framework must evolve to support the long term evolving needs of an organization.

3 Summary

When using object-oriented frameworks, one of the key decisions that needs to be made is whether or not the framework fits the application. Since frameworks can be complex, gaining a deep enough understanding of the framework to make that decision often requires the time consuming process of actually using the framework. Capturing information about the applicable domain of the framework is a way to ease this decision. Limitations and design tradeoffs about the framework can help to show what the framework cannot be used for. Hook descriptions can be used to show what the framework can be used for. In the middle is a region of uncertainty that will exist for all frameworks, but tool support and documentation can aid in determining the suitability of the framework.

References

[1] K. Beck and R. Johnson. *Patterns Generate Architectures.* In Proceedings of ECOOP'94, Bologna, Italy, 1994, 139-149.

[2] M. Fayad and D. Schmidt. *Object-Oriented Application Frameworks.* CACM, 40(10), October 1997.

[3] Froehlich, G., Hoover, H.J., Liu, L. and Sorenson, P. 1997. Hooking into Object-Oriented Application Frameworks. *Proceedings of the 1997 International Conference on Software Engineering.* pp. 491-501.

[4] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA, 1995.

[5] D. Gangopadhyay and S. Mitra. *Understanding Frameworks by Exploration of Exemplars.* In Proceedings of 7th International Workshop on Computer Aided Software Engineering (CASE-95), Toronto, Canada, 1995, 90-99.

[6] R. Johnson. *Documenting Frameworks Using Patterns.* In Proceedings of OOPSLA'92, Vancouver, Canada, 1992, 63-76.

[7] G. E. Krasner and S. T. Pope. *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80.* Journal of Object-Oriented Programming, 1(3), August-September 1988, 26-49.

[8] W. Pree. *Design Patterns for Object-Oriented Software Development.* Addison-Wesley Publishing Company, Reading, MA. 1995