Object-Oriented Framework-based Software Development: Problems and Experiences

Jan Bosch
Peter Molin
Michael Mattsson
and
PerOlof Bengtsson

University of Karlskrona/Ronneby, Department of Computer Science and Business Administration

The claimed advantages of object-oriented frameworks are, among others, increased reusability and reduced time to market for applications. Although several examples have shown these advantages to exist, there are problems and hindrances associated with frameworks that may not appear before their usage in real projects. In this paper a number of problems related to frameworks are described organised according to four categories, i.e. framework development, usage, composition and maintenance. For each category, the most relevant problems and experiences are presented.

Categories and Subject Descriptors: D.1.5 [**Object-oriented Programming**]: Frameworks

General Terms: Software reuse, Object-oriented frameworks

1. INTRODUCTION

The notion of object-oriented frameworks, after its conception at the end of the 1980s, has attracted attention from many researchers and software engineers. Frameworks have been defined for a large variety of domains. In addition to the intuitive appeal of the framework concept and its simplicity from an abstract perspective, experience has shown that framework projects contribute to increased reusability and decreased development effort: see e.g. [Moser and Nierstrasz 1993]. The authors of this paper have been involved in the design, maintenance and usage of a number of object-oriented frameworks, among others for fire-alarm systems [Molin 1996; Molin and Ohlsson 1996], measurement systems [Bosch 1998], gateway-billing systems [Lundberg 1996], resource allocation systems and systems for process operation [Betlem et al. 1995]. During these framework related projects several obstacles were identified that complicated the use of frameworks or diminished their benefits.

The identified obstacles can be organised into four categories and are discussed in the following sections. In section 2, issues related to framework development are described, i.e. the framework until it is released and used in the first real application. Section 3 discusses the problems of instantiating a framework and application

Address: Soft Center, S-372 25 Ronneby, Sweden. E-mail: Jan.Bosch@ide.hk-r.se

development based on a framework. The composition of multiple frameworks into an application or system and the composition of legacy code with a framework are discussed in section 4 whereas the evolution of a framework over time, starting with the initial framework design and continuing with the subsequent framework versions is discussed in section 5. The paper is concluded in section 6.

2. FRAMEWORK DEVELOPMENT

The development of a framework is different from the development of a standard application. The important distinction is that the framework has to cover all relevant concepts in a domain, whereas an application only is concerned with the application requirements. Some problems that may be encountered during framework development are presented here:

—**Business models**: Even though it may be well feasible to develop a framework for a particular domain from a technological perspective, it is not necessarily advantageous from a business perspective. The return on investment from a developed framework may come from selling the framework to other companies, but it often, to a large extent, relies on future savings in development effort within the company itself, such as higher software quality and shorter lead times. One of the main problems for the management of software development organisations or departments, is that to the best of our knowledge, no reliable business models for framework development exist.

—**Verifying abstract behaviour**: Since a framework can be used in many different, on forehand unknown ways, assessing the correctness and versatility of the functionality provided by the framework is generally very hard. Since the framework often defines abstract behaviour it is not possible to completely test the framework before it is instantiated.

—**Framework release problem**: Releasing a framework for application development requires it to fulfil criteria related to stability, reusability, flexibility and documentation. As for reusability, several authors have addressed this issue and in [Poulin 1994] several of the proposed approaches are compared. Deciding whether the framework is sufficiently well documented is hard, since there exists no generally accepted documentation method that covers all aspects of the framework, but also because it is difficult to determine whether the documentation is understandable for the intended users.

3. FRAMEWORK USAGE

Although it is feasible to produce complex applications based on a framework with rather modest development effort, we have experienced a number of problems associated with the usage of a framework. When using a white-box framework, but even when using a black-box framework, it is necessary to understand the concepts and architectural style of the framework in order to develop applications that conform to the framework. The framework understanding is a pre-requisite for evaluating the applicability of a framework, the amount of required adaptation and how the adaptation can be carried out. Since framework documentation approaches suffer from several problems it is often difficult to obtain an understanding of the framework. Some of the usage problems are:

—**Applicability of the framework**: Deciding whether a particular framework is suitable for an application is hard without spending considerable effort on evaluation. Obviously, the framework should cover the domain required by the application, but also other dimensions play a role. For instance, the driving quality requirements of the framework should match those of the application. Also, the context required by the framework is an important dimension. In case only a partial match exists, the potential framework user has to decide whether it could be worthwhile to redesign parts of the framework.

—**Estimations of the application development**: As mentioned, understanding a framework requires considerable effort. From a productivity point of view, framework-based application may seem slow compared to a traditional approach. However, complex applications can be build very fast so efficiency generally is high. The consequences of traditional estimations techniques based on number of produced lines of code is inadequate in the framework case [Moser and Nierstrasz 1993]. A second problem is the sensitivity of estimates of the amount of work required for a specific application. It can be difficult to foresee if a specific requirement is supported by the framework. If it isn't, a potential implementation can mismatch the intentions of the framework designers possibly causing the resulting effort to be even larger than using a traditional approach.

—**Debugging the application**: Traditional debuggers have problems when debugging programs using libraries. Frameworks, and especially black-box frameworks, have the same problem as libraries. Furthermore, since frameworks often are based on the 'hollywood principle' [Sparks et al. 1996], the problems are even more difficult. It can be very difficult to follow a thread of execution which mostly is buried under framework code.

4. FRAMEWORK COMPOSITION

Increasingly often, a framework needs to be composed with other frameworks or with reusable legacy components. Composing frameworks, however, may lead to a number of problems since frameworks generally are designed based on the traditional perspective where the framework is in full control.

—**Architectural mismatch**: The composition of two or more frameworks that seem to match at first may prove to be much more difficult than expected. One reason is referred to as the architectural mismatch [Garlan et al. 1995], i.e. the architectural styles based on which the frameworks are designed are different, complicating composition so much that it may be impossible to compose. Explicitly specifying architectural decissions, such as the style underlying a framework design, seems to be the first step towards a solution to this problem.

—**Overlap of framework entities**: Two frameworks may contain a representation (e.g. a class) of the same real-world entity, but modelled from their respective perspectives. In the application, however, the real-world entity should be modelled by a single object and the two representations should be integrated into one. Possible solutions are to use multiple inheritance or aggregation. Another approach that does not require changes in the frameworks is to subclass each framework class that represent the real-world entity and aggregate the two subclasses into one class. However, not all cases are covered by these solutions.

—**Composition of framework control**: A potential problem when composing two calling [Sparks et al. 1996] frameworks is that both frameworks expect to be the controlling entity in the application and in control of the main event loop. Possible solutions are to give each framework its own thread of control, to encapsulate each framework by a wrapper intercepting all messages sent to and by the framework or to remove the relevant parts of the control loops from the frameworks and to write an application specific control loop. However, non of these solutions deals with the situation where internal framework events need to be exported.

—**Composition with legacy components**: A software engineer may want to include existing, legacy, classes that need to be integrated with the framework. Framework classes generally contains behaviour for internal framework functionality in addition to the domain specific behaviour. Since the legacy component only contains domain functionality, it is generally hard to integrate legacy components with a framework. Existing solution approaches include the use of the *Adapter* design pattern [Gamma et al. 1995], replacing framework classes or parts of classes with legacy code, provided source code is available, or designing the framework based on the role metaphor [Lundberg and Mattsson 1996].

—**Framework gap**: Two frameworks that are composed may still fail to fulfil an application's requirements. This is referred to as the *framework gap* problem [Sparks et al. 1996]. Possible solutions include wrapping, the use of mediating software and redesign of the frameworks.

—**Composition of framework functionality**: Sometimes a real-world entity's functionality has to be modelled through composition with parts of functionality from different frameworks. Consider the case of a software structure with three layers, i.e., a three-tier architecture, where each layer is represented by a framework. A real-world entity has to be represented in each of the frameworks, since each framework described one aspect of its functionality. There are several solutions addressing the simples cases such as composing the framework classes with the required functionality into a single class using aggregation or multiple inheritance. Another approach is to extend the application domain specific class with notification behaviour, e.g the *Observer* design pattern [Gamma et al. 1995].

5. FRAMEWORK MAINTENANCE

Development of a framework has to be seen as a long term investment and, as such, it has to be seen as a product that needs to be maintained. In the beginning of a framework development effort often several design iterations are necessary. When a change has to take place in a framework either because of an error or because of some required enhancement, the following maintenance problem occurs:

—**Choice of maintenance strategy**: Given the situation that the framework has changed, either because the domain covered by the framework was incomplete or the application mismatches the framework's domain, it is necessary to decide whether to redesign the framework or implement a work-around for this specific application. In the case where we decide to redesign the framework, the application development must be delayed until a new version of the framework is available. In addition, the organisation is forced to maintain two versions of

the framework since there may exist applications based on the old version of the framework. In the case of a work-around in the application, the maintenance problem will instead occur for the developed application. This maintenance strategy will not be suitable if the application under development will have a long expected life-time. However, this may be an acceptable situation if it is expected that no similar applications will be developed in the foreseeable future.

—**Business domain change**: Frameworks are generally developed in a domain that is closely related to the organisations business domain. However, the domain is often weakly defined and evolves over time. Consequently, the domain captured by the framework has to be adapted to follow this change and this affects the existing framework. The probability of business domain change is an important risk factor that has to be considered in the investment of the framework development effort. There are, in principle, three approaches attacking the problem of business doming change. One may define the original framework domain much wider than currently useful, some kind of super-domain, that will capture most future new domain changes. Another approach is to handle the business domain change problem by redesigning the framework such that it covers both the original domain and the new domain. A third approach is to reuse ideas from the original framework and develop a framework for the new business domain. All these approaches have disadvantages.

6. CONCLUSION

Object-oriented frameworks provide an important step forward in the development of large reusable components, when compared to the traditional approaches. However, as we report in this paper, there still exist a number of problems and hindrances that complicate the development and usage of object-oriented frameworks. These problems can be divided into four categories:

—**Framework development**: Problems in the development of a framework are related to the lack of business models, framework verification and releasing.

—**Framework usage**: The user of a framework has problems related to deciding applicability, estimations of development time and size of application specific code and debugging.

—**Framework composition**: In case the application requires multiple frameworks to be composed, the software engineer may experience problems with respect to mismatches in the architectural styles underlying the frameworks, overlap of framework entities, possible collisions between the control flows in calling frameworks, the integration of legacy components, framework gaps and composition of entity functionality.

—**Framework maintenance**: Being a long-lived entity, frameworks evolve over time and need to be maintained. The framework development team may experience problems with choosing the appropriate maintenance strategy and incorporating business domain changes.

The problems reported in this paper we believe to be relevant both for software engineers and researchers on object-oriented reuse. After reading this paper, practitioners will hopefully be able to avoid problems that some of us experienced. In

addition, we believe that this paper could be used as a research agenda for researchers on object-oriented software development or on software reuse in general.

REFERENCES

Betlem, B. H., van Aggele, R. M., Bosch, J., and Rijnsdorp, J. E. 1995. An object-oriented framework for process operation. Technical report, University of Twente, Enschede, Netherlands.

Bosch, J. 1998. Design of an object-oriented framework for measurement systems. In M. Fayad, D. Schmidt, and R. Johnsson Eds., *Object-Oriented Application Frameworks* (April 1998).

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

Garlan, D., Allen, R., and Ockerbloom, J. 1995. Architectural mismatch or why it's so hard to build systems out of existing parts. In *Proceedings of the 17th International Conference on Software Engineering* (April 1995).

Lundberg, C. and Mattsson, M. 1996. On using legacy software components with object-oriented frameworks. In *Proceedings of Systemarkitekturer '96* (Boras, Sweden, 1996).

Lundberg, L. 1996. Multiprocessor performance evaluation of billing gateway systems for telecommunication applications. In *Proceedings of the ICSA conference on Parallel and Distributed Computing Systems* (September 1996), pp. 225–237.

Molin, P. 1996. Verifying framework-based applications by conformance and composability constraints. Research Report 18/96 (Feb.), University of Karlskrona/Ronneby, Ronneby, Sweden.

Molin, P. and Ohlsson, L. 1996. Points & deviations -a pattern language for fire alarm systems. In *Proceedings of the 3rd International Conference on Pattern Languages for Programming* (September 1996). Monticello IL, USA.

Moser, S. and Nierstrasz, O. 1993. The effect of object-oriented frameworks on developer productivity. *IEEE Computer*, 45–51.

Poulin, J. 1994. Measuring software reusability. In *Proceedings International Conference on Software Reuse* (November 1994).

Sparks, S., Benner, K., and Faris, C. 1996. Managing object-oriented framework reuse. *IEEE Computer*, 53–61.