Enterprise Frameworks: Guidelines for Selection

Mohamed E. Fayad, University of Nebraska, Lincoln
David S. Hamu, TRW
fayad@cse.unl.edu, dhamu@acm.org

An Enterprise Framework (EF) is a software architecture. Such frameworks expose a rich set of semantics and modeling paradigms for developing and extending enterprise applications. EFs are, by design, the cornerstone of an organization’s systems development activities. EFs offer a streamlined and flexible alternative to traditional tools and applications which feature numerous point solutions integrated into complex and often inflexible environments. Enterprise Frameworks play an important role since they allow reuse of design knowledge and offer techniques for creating reference models and scalable architectures for enterprise integration. These models and architectures are sufficiently flexible and powerful to be used at multiple levels, e.g. from the integration of the planning systems of geographically distributed factories, to generate a global virtual factory, down to the monitoring and control system for a single production cell. These frameworks implement or enforce well-documented standards for component integration and collaboration. The architecture of an Enterprise framework provides for ready integration with new or existing components. It defines how these components must interact with the framework and how objects will collaborate. In addition, it defines how developers' work together to develop and extend enterprise applications based on the framework. Therefore, the goal of an Enterprise framework is to reduce complexity and lifecycle costs of enterprise systems, while ensuring flexibility.

Key Words and Phrases: Enterprise Frameworks, Software Architecture, Distributed Computing, Object-Orientation, Aspect-Oriented Frameworks, Software Stability, Extensibility, Customization, and framework economics.

-----------

Author's address: Computer Science and Engineering, University of Nebraska, 108 Ferguson Hall, Lincoln, NE 68588-0115; e-mail: fayad@cse.unl.edu.
Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2000 ACM 00360-0300/00/0300es
A TYPICAL FRAMEWORK

Figure 1 depicts a portion of a framework from a sales order processing perspective. Functionally identical to the OMA Reference Model [Ben-Natan 1995], clients (or application objects) are developed to solve domain-specific problems. These objects make use of the Distributed Objects Transport and Object Services. In the example, a sales agent, while processing orders interacts with the framework through the sales processing client. The client retrieves workflow instructions via the Workflow Services, which triggers any data collection screens and provides relevant product availability and pricing lookups.

While sequencing through the workflow, the Sales Processing Client may also be prompted with information about special sales promotions retrieved via the Marketing Services.

ENTERPRISE FRAMEWORK CHARACTERISTICS, CHALLENGES, AND CRITERIA

Framework requirements are generally defined by an independent standards body (e.g., CORBA), a Software Vendor (e.g. Microsoft’s MFC, COM and DCOM, FASTech’s FACTORYworks, Taligent’s CommonPoint) or a Systems Engineering Group (e.g., Motorola’s CIM Baseline.) Frameworks combine the best features of state-of-the-art programming languages, development environments, and tools. In addition, frameworks provide an extensive library of business objects supporting the intended application domain [Fayad et al. 2000, Hamu-Fayad 1998].

Our studies of enterprise frameworks suggests the following characteristics, challenges, and criteria are commonly exposed in an Object-Oriented Enterprise Framework:

1. Mature run-time functionality
2. Support for extensibility, tailorability, and customizability
3. A catalog of business objects and enduring business themes (EBTs)
4. A workflow management metaphor and enduring business processes (EBPs)
5. Accomplishing software stability
6. A model for distributed objects
7. Support integration of multiple application frameworks and legacy components
8. Platform independence or portability
9. Mature framework documentation
10. Support for role object pattern and ease of use
11. Web- and E-business-ready
12. Support for separation of concerns
13. Sound investment – framework economics
14. Framework adequacies

Each of these characteristics, challenges and criteria is described in detail in [Fayad et al. 2000, Hamu 1999]

THE MAKE VS. BUY DECISION

Although some very successful enterprise frameworks may lack any number of the characteristics described above, a good framework will generally incorporate most if not all of these characteristics. In fact, we recommend evaluating frameworks against each of these characteristics during the selection process.

Many organizations undertake the costs of developing application frameworks without first analyzing their need for acquiring a framework, without documenting the functional requirements of the framework, and without evaluating commercially available products. Still other organizations may select and purchase a commercially available framework without a thorough analysis.

Deciding whether to make or buy a framework is a fundamental question for an organization. Unlike other software selection decisions, the selection of a framework is a decision that an organization must live with for 10 to 15 years, perhaps longer. The long life-span of frameworks relative to that of legacy applications or traditional development tools is due to the fact that frameworks are more flexible and adaptable to changing needs. Therefore, frameworks are less susceptible to obsolescence than legacy products. However, the long life-span and rich functionality of frameworks does not come without costs, namely:

- Application frameworks require a considerable up-front cost vs. traditional applications or application development tools. This is due to the comprehensive development environment and the extensive set of domain-specific objects which are delivered with the framework.

- The investment in framework training and staff development is significantly higher and more specialized than traditional application development platforms.

Therefore, this decision should not be made without first completing a thorough analysis of the business reasons for acquiring a framework. Moreover, it is essential that the selection process be applied to both make and buy options. Even when you are inclined to build a framework, other frameworks should be researched.

In short, failing to plan is planning to fail. A rigorous systems engineering approach must be applied to framework projects as with any software project. Sound system analysis encompasses the following activities [Pressman 1987]:

With frameworks, it is recommended that a framework team be formed to conduct the analysis. Suggested by Goldberg and Rubin [Goldberg-Rubin 1995], a framework team is similar to the Joint Application Design (JAD) Team described by Martin [Martin 1991], and consists of the following resources:

- Executive Owner / Business Area Expert
- Project Manager / Systems Analyst
- Framework Designer/Evaluator
- Component Evaluators
- Object Expert/Coach

The first four analysis activities are generally completed by the project manager or systems analyst with input from the other members of the framework team. The evaluation of alternatives (also called the selection process) is performed jointly by all members of the framework team. In the case of frameworks for business applications, this team must include at least one representative of the end-user community; namely, the Business Area Expert.

A clear understanding of the need for the framework is the first step in the analysis. High level functional requirements are then defined in during the feasibility study. These requirements along with the Economic Analysis and Evaluation of Alternatives provide the necessary inputs for the Make vs. Buy decision.

**DECISION FACTORS**

Other selection criteria which should be considered include supplier stability and support or service offerings. The latter are important, because when you buy a framework, you have effectively outsourced a portion of your software development. The framework team will not be responsible for support and maintenance except to the extent that they are developing extensions or plug-in components for the framework.

**ADVANTAGES AND DISADVANTAGES OF PURCHASING A FRAMEWORK**

Frequently the final selection comes down to a make vs. buy decision. Therefore, it is important to understand the relative advantages and disadvantages of these options. The primary advantage of purchasing a framework is that the development of commercial products or in-house applications can begin almost immediately. This is particularly true when time-to-market and strategic advantage are key concerns. Framework vendors are generally be concerned about providing wide domain coverage and flexible modeling tools within the scope of their framework. These feature must be significantly broad to attract a majority of the potential customers in the target marketplace otherwise the product will be difficult to sell. Therefore, a purchased framework is likely to be sufficiently flexible for most customers in the target market and is likely to provide wide coverage of the characteristics of frameworks described above.
In addition, purchased frameworks promise higher quality because of the demands of a wide customer base and the fact that commercial frameworks will have successfully completed a lengthy beta software program. Maintenance is easier for a purchased framework, since the vendor will generally provide a technical support desk to field maintenance requests and trouble reports. Lifecycle costs are reduced as a result of the wide domain coverage and flexibility of the framework [FASTech 1995]. Scalability and adaptability are increased because you can add new objects to the system without modifying existing objects [Fayad et al. 1999a, 1999b, 1999c].

When building a framework, the process of designing and developing the framework may require a team of developers working from one to three years depending on the features supported by the framework. Additional time will be invested trouble-shooting the framework. Whereas, a framework vendor has already completed development of a commercial product and development and even trouble-shooting costs are distributed across the vendor’s entire customer base.

However, purchasing a framework is not always appropriate. Immature frameworks may not provide adequate coverage of the decision factors described above. This is particularly true in very specialized domains where the market potential for frameworks is limited. In such domains, it is generally advisable to build a framework. In fact, if the framework development effort is successful, an opportunity for commercialization of the newly built framework may present itself (although this prospect should not be factored into the make vs. buy decision, because it is a high-risk assumption.)

The lack of fully adopted standards for frameworks also contributes to increased risk for developing a framework in house. Speed of execution can be an issue. Since object services may be very complex, there is often a risk of performance bottlenecks. Availability of qualified developers is also an important consideration and it is essential to obtain and evaluate the list of available consultants or partners working with the vendor company who have experience in development and deployment with the vendor’s framework.

Due to these considerations, it is difficult to estimate the cost of building an enterprise framework. Therefore, all other things being equal, in a domain with strong market opportunities and strong offerings from framework vendors, it is better to purchase a framework than to build.

CONCLUSIONS

This paper has presented detailed guidelines for selection of Object-Oriented Enterprise Frameworks. We have offered our recommendations for the make vs. buy decision as it applies to frameworks. We have shown that good Object-Oriented Enterprise Frameworks are characterized by support for the essential tools used in systems analysis, modeling, design, development, testing, maintenance, and administration. Therefore, although this is an emerging area in software engineering, it reflects the natural evolution in the development of software engineering tools and practice.

Although we recommend purchasing vs. building these frameworks, it is also understood that this decision needs to be made in context and with the application of a formal systems analysis. While commercial software vendors have made fantastic strides in reflecting cutting edge research in products delivered to market, software engineers need to evaluate risk when selecting frameworks.
Project managers will find very reliable and mature product offerings which are not truly object-oriented, because they are built on relational rather than object databases and use non-object-based distribution mechanisms. This highlights that there is not yet widespread adoption of CORBA, DSOM or DCOM, and the fact that Relational Database Management Systems still dominate the commercial market for software products and tools. Nevertheless, it is likely to be cost effective to select a mature framework provider with an object-oriented strategy and roadmap even if the product is not fully object-oriented today.

While the notion is in its infancy today, it is our belief that Object-Oriented Enterprise Frameworks will be at the core of software engineering technology in the twenty-first century.

REFERENCES


Goldberg, Adele and Rubin, Kenneth S., Succeeding with Objects: Decision Frameworks for Project Management, Addison Wesley, 1995


Martin, James, Rapid Application Development, McMillan Publishing Company 1991