JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 1, January-February 2004

The Elusive Search for Business Frameworks

Meta Thoughts on Building Domain Oriented Frameworks

Dave Thomas,
Bedarra Corp., Carleton University and University of Queensland

1 THE PROMISE OF BUSINESS FRAMEWORKS

Object and Component [1] technology promised frameworks that would enable businesses to achieve substantial reuse. Such reuse reduces the time and expense of development and increases the quality by building on tested software. Business frameworks represent portions of the business, typically a specific domain such as retail banking, financial instruments, insurance, and human resources. Using a business framework, developers can assemble a complete business application by tailoring the framework for a particular business. They represent the Holy Grail in the value chain of frameworks and components.

The ultimate goal of an ideal business component is a Business Franchise. A franchise would consist of a unified business process for that specific business, a set of human and machine actors (job descriptions) and a set of frameworks tailored for a specific franchise instance.

Some Assembly Required

While there has been much written about business frameworks [5] [6] and their benefits, in reality there are relatively few successful examples. Instead there have been several monumental failures and several collective efforts that have come to a standstill. Indeed, despite efforts in many industries, both OO (OMG [7], ACCORD [5]) and more recently XML (OASIS, ACCORD) have had great difficulty defining business objects.

The challenges are not only of a technical nature. There is strong resistance to reuse in project driven organizations and in many cases financial disincentives to reducing manpower on an assignment through the introduction of reusable code. Despite substantial efforts for COTS[1] by the US government, the COTS reality is still far from

[1] Commercial Off The Shelf

Cite this column as follows: Dave Thomas: "The Elusive Search for Business Frameworks", in *Journal of Object Technology*, vol. 3, no. 1, January-February 2004, pp. 7-13.
http://www.jot.fm/issues/issue_2004_01/column1

what it should be, especially in services where reducing the number of developers is not in the best interest of the service provider.

The Markitecture Approach – *Hey Mister Want to Buy A Framework?*

Some so-called business frameworks are really just application code that is typically massively customized by consultants to solve a business problem. It is common practice to cite a framework or COTS component during a bid, yet after it has been delivered there is no way an updated version of the COTS code can be plugged into the same application. Effectively this is cut, paste, and edit reuse rather than systematic reuse.

Open source provides an increasing number of technical frameworks, many with commercial grade quality such as Apache and JBOSS, as evidenced by their adoption by major vendors. The challenge in using such frameworks is the planned evolution of the software as the frameworks evolve. All too often, white box frameworks are modified and combined with other frameworks creating a custom solution with its associated dependency on the supplier. One needs to remember that major computer vendors provided the source for their operating systems in the past and it was their customers who, having made extensive modifications to the source code, were unable to upgrade to new versions of the operating system and compilers and were held hostage to the Y2K problem.

Unfortunately, the situation is clouded even further by the inconsistent use of the term “framework” in Enterprise Framework and even lawyers talk about contract frameworks. Neither of these examples have any relationship to an OO framework.

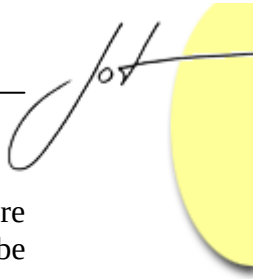
Applications Frameworks Are Not Business Frameworks

Many often cited examples are really just technology frameworks used in business applications that have little or no domain knowledge. For example J2EE, .NET, CMS, eBusiness etc. are often called ***Application Frameworks*** [3] but in reality they are the infrastructure on which business applications/frameworks are constructed and contain very little, if any, application domain knowledge. They provide containers for business objects when augmented with patterns for constructing applications [8] [11].

2 BUSINESS FRAMEWORKS VERSUS TECHNOLOGY FRAMEWORKS AND PRODUCT LINES

Business Classes – A Little Reflection

There is a lot of misinformation about business frameworks, their development and deployment. Most failures are attributed to the code bulk, complexity and lack of robustness of the framework from application to application. Their critics often conclude that the framework design team lacked the proper engineering skills or the proper domain knowledge. It is well known that framework engineering requires a great deal of expertise



and clearly the bulk and complexity of many frameworks can be improved by more experienced designers. Similarly, there is a critical need for the domain specialists to be well versed in the business domain [13] [14].

However we believe that the problem goes beyond the lack of domain knowledge and framework design expertise. We argue that there is an intrinsic difference between technology frameworks and product lines and business frameworks. This difference means that business frameworks need to be constructed with a reflective engineering [12] [3] [18] approach rather than by using concrete classes and classical analysis.

Technology Frameworks

Technology frameworks consist of a base class library usually provided with the OO language environment and the classes to address a specific technology requirement such as persistence, security, logging, XML, network communications, MVC, Portals etc. Today we often consider such frameworks to be part of the *platform* on which the application will execute. The classes are concrete programming abstractions often based on known standards and known patterns of best practice in the software community. There are numerous examples of such frameworks and they are in pervasive use throughout the industry.

Product Line Architectures

Product Line Architectures [2] consist of product-centric class libraries (e.g. waveforms) and associated frameworks that allow for several product variants with different features. In some companies and industries, there are clear families of software components that have a direct analogy to the products of the industry. Such industries have strong manufacturing/engineering cultures and a natural affinity and acceptance of the concept of a component and its value in product development. There are many successful examples of different product variants, manufactured from the same code base reusing substantial portions (> 70%) of the product line framework.

Business Frameworks

Both technology and product line frameworks have well known and relatively well understood artefacts that can be readily mapped to concrete classes. Like technology frameworks, business frameworks consist of core domain libraries that define the concrete artefacts of a given domain. For example Money [16] [9], Country, Date and Time are rich and challenging core classes needed for all business applications. Similarly a point of sale library would contain SKUs, PLUs[2], CreditCards etc. that one would find in every retail application. Designing such libraries is often more challenging [16] than designing a red black tree in computer science.

[2] SKU – Stock Keeping Unit, PLU = Price Look Up

The Challenge of Generic Domain Objects

Unlike technology frameworks, business frameworks make extensive use of generic domain objects such as Customer, Account, Order, Invoice, Policy etc. Naive object model approaches lead to concrete classes such as customer, order, account, invoice etc. and their associated relationships represented in a UML model. Unfortunately, this isn't sufficient for rich domain objects. Martin Fowler [15] illustrates the failure of the naïve approach in his person (patient) example. They contain too much variability to be easily represented using naïve concrete class models. This approach has in a large part stymied efforts on object models (OMG, ACCORD) and XML models (OASIS, ACCORD, UBL).

Naïve business object models yield concrete class definitions for Customer, Order, and Invoice etc. however inevitably these naïve models prove themselves to be inadequate for wide scale deployment. All too frequently these frameworks need to be extensively modified in each customer setting. While this may reduce the development time or expense, it will then be impossible to upgrade the customer to a new version of the framework.

3 REFLECTIVE ENGINEERING FOR BUILDING DOMAIN CLASSES

We believe that there is considerable promise by approaching business frameworks using a meta modeling approach we have called MDD. Briefly we describe the key implementation techniques that are based on reflective engineering [12]. Reflective engineering includes a variety of techniques – Meta Programming, Table Driven Programming, Generative Programming [3], Aspect Oriented Software Development [12], Multi Dimensions of Concern [12] and Feature Oriented Programming [18]. Each if these approaches programmatically compose code fragments to produce the resulting program instance.

Model Driven Components - The Business Class Factory

This form of model driven development is now in vogue with the OMG MDA, however we prefer to call this MDC (model driven components). It is important to note that the modeling need not be done in UML or MDA style, although UML is in practice useful for at least portions of the business meta model.

Class Factories provide a mechanism for coping with the variations too complex to capture in concrete classes. Concrete classes are created from models (meta models). This provides a solution for managing the variations across platforms (Java, C#, VB.NET, Python...), business (Home Depot, Canadian Tire), and country (UK, France, China). They provide computational machinery to construct tailored concrete classes for a specific framework deployment.

Customer objects, for example, can consist of hundreds of attributes depending on the industry and the nature of the information retained about customers. A CustomerClass meta class encapsulates the complex variety of customer attributes and using a rule driven wizard, synthesizes a specific concrete class to suit the needs of a given project or business.

In a modern integrated enterprise, customer's instances are materialized in the middle tier from several different customer systems operated by different companies. A Swiss bank, for example, may now own another Swiss Bank, as well as a bank in Austria and Germany. If a customer has accounts in any of the original banks, they expect access to all of their bank accounts since in their view they are all accounts of the Swiss Bank. In practice, the bank's IT is often run separately using the existing systems for various business and technical reasons. In order to provide a common retail bank customer view, the bank must assemble the customer instance on the fly from each of the banks.

Such federated business objects are today the norm rather than the exception. Our CustomerClass factory, for example, would contain the essential behaviour to interface to various schemas and repositories such that the installer of the framework could select the elements from the respective Swiss, German, and Austrian bank schemas to construct the appropriate federated business object. The need to build tailored, federated business objects is a common meta pattern in business frameworks.

4 COPING WITH BEHAVIORAL VARIABILITY

Thin Frameworks Are More Malleable

It is well known that shallow class hierarchies are much better than deep hierarchies. Specialization should only happen low in the hierarchy where it is anticipated and planned for. For example, in a financial instruments framework, the class hierarchy swaps, options etc. are a powerful descriptive mechanism. However, it is easy to go way overboard building such a class library that results in a system that is difficult to maintain and has poor performance. Experience has shown that such instrument frameworks are best realized with a thin class hierarchy, which uses efficient array operations to implement the specific time series behaviors [17].

Policy Driven Behavior

A proven way to deal with the variety of complex behaviors is a technique often referred to as *policy driven behavior*. The great advantage of a policy approach is that often the behavior can be defined and realized by a business analyst or end user. In many cases, the software can be modified on the fly at run-time. This will be increasingly important as businesses move to be real-time enterprises.

A policy driven system uses a set of *"rules"* to describe a specific policy and a generic algorithm to evaluate the policy. In the past this was often called *table driven programming* and spoken of as a best practice. Examples of techniques include decision

tables (policy), constraints (spreadsheets), state machines (workflow), pure functional behavior (data/message flow) to more exotic expert systems and logic programming.

Business Engineering - A Challenge and Opportunity

In this article we explained why business frameworks remain challenging to build and deploy. We touched briefly on some techniques for coping with the rich variety and complexity of business frameworks. However there is a need for business engineers to define rich domain meta models as well as implement these models. We are just beginning to be able to describe business semantics. The recent work on ontologies, business process engineering, workflow, organizational design and business components is all-promising but it is far from complete.

One only needs to see the immense gaps between the executive model charts provided by the management consulting firms, the business process models of the BPM of the process reengineering consultants and the UML/XML models of the IT industry. There is a need for a comprehensive understanding both at a practical and research level for business semantics and for a well-developed field one might call business engineering.

Business Frameworks/Components requires the combined skills of framework engineering, domain analysis and business engineering. Unfortunately there are few if any DBAs or PHDs in this area, and only a few practitioners with the breadth and depth to cope with the huge shortage in this field.

REFERENCES

[1] Clemens Szyperski. *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley, 2nd edition 2002

[2] Paul Clements, Linda Northrop. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001

[3] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming - Methods, Tools, and Applications*, Addison-Wesley, June 2000 http://www.cs.rice.edu/~taha/gpce/

[4] Mohamed E. Fayad. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, John Wiley and Sons, 1999 Annotated References, http://www.cse.unl.edu/~fayad/Books/ImplemB2/refs.htm

[5] http://www.castek.com/PDF/LessonsLearned.pdf

[6] IBM (Ed.) (1998): San Francisco Project Technical Summary. http://www.ibm.com/Java/Sanfrancisco/ prd_summary.html. 16.06.1998

[7] OMG (Ed.) (1996): Common Facilities RFP-4: Common Business Objects and Business Object Facility. ftp://ftp.omg.org/pub/docs/cf/96-01-04.pdf. 21.01.1998

[8] Peter Herzum and Oliver Sims. *The Business Component Factory*, John Wiley and Sons, 1999 http://www.componentfactory.org/default.htm

[9] Kent Beck. *Test Driven Development: By Example*, Addison-Wesley, 2002

[10] Martin Fowler. *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002

[11] Deepak Alur, John Crupi and Dan Malks. *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall / Sun Microsystems Press, 2nd edition, June 2003

[12] Dave Thomas. "Reflective Software Engineering - From MOPS to AOSD", in *Journal of Object Technology*, vol. 1, no. 4, September-October 2002, pp. 17-26. http://www.jot.fm/issues/issue_2002_09/column1

[13] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, 2003

[14] Domain Analysis and Modeling http://www.iturls.com/English/SoftwareEngineering/SE_mod5.asp

[15] Martin Fowler. *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1996

[16] Ward Cunningham, John Cunningham and Nick Knowles, Personal communications.

[17] Greg Baster, Personal Communication.

[18] D. Batory, J.N. Sarvela, and A. Rauschmayer, "Scaling Step-Wise Refinement", *International Conference on Software Engineering* (ICSE-2003).

About the author

Dave Thomas is CEO of Bedarra Corp., Adjunct Professor at Carleton University, Canada and University of Queensland, Australia, founding Director of AgileAlliance.com, and founder of Object Technology International. Bedarra works with research labs and commercial partners to transition innovations into products and practices.