IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITP4389

Achieving Effective Software Reuse for Business Systems

Daniel Brandon, Jr.
Christian Brothers University, Information Technology Management Department
650 East Parkway South, Memphis, TN 38104, (dbrandon@cbu.edu, phone: 901.321.3615, fax: 901.321.3566)

OVERVIEW

"Reuse [software] engineering is a process where a technology asset is designed and developed following architectural principles, and with the intent of being reused in the future" [Bean, 1999]. "If programming has a Holy Grail, wide-spread code reuse is it with a bullet. While IT has made and continues to make laudable progress in our reuse, we never seem to make great strides in this area" [Grinzo, 1998]. The quest for that Holy Grail has taken many developers over many years down unproductive paths" [Bowen, 1997]

This paper reports on software reuse research (both literature research and design/coding research) and presents an approach for effective software reuse in the development of business systems. This approach is based on Object Oriented technology and provides for both the specification and enforcement of software reuse and corporate standards.

BUSINESS SYSTEMS

Business software systems are typically composed of three logical portions or layers as shown in Figure 1. The "presentation layer" involves the primary user interaction typically via a graphical user interface (GUI). The "business logic" layer provides database connectivity, validation, security, transaction control, and other sequencing or optimization control. This layer may be packaged by a vendor in an application or transaction server or written by a user. The "database layer" provides for the manipulation of persistent data, which for most business systems today is stored in a relational database. The interface to this process is a well defined standard application programming interface (API) like ODBC or JDBC using SQL.

Figure 1

NEED FOR REUSE

Today's software development is characterized by many disturbing but well documented facts, including:

- Most software development projects "fail" (60% [Williamson, 1997])
- The supply of qualified IT professionals is much less than the demand
- The complexity of software is constantly increasing
- IT needs "better", "cheaper", "faster" software development methods

"Object technology promises a way to deliver cost-effective, high quality and flexible systems on time to the customer "[McClure, 1996]. "IS shops that institute component-based software development reduce failure, embrace efficiency and augment the bottom line" [Williamson, 1997]. "The bottom line is this: while it takes time for reuse to settle into an organization – and for an organization to settle on reuse – you can add increasing value throughout the process". [Barrett, 1999] We say "object technology" not just adopting an object oriented language (such as C++ or Java), since one can still build poor, non object oriented, and non reusable software even using a fully object oriented language.

TYPES AND APPLICATIONS OF REUSE

Radding defines several different types of reusable components [Radding, 1998]:

GUI widgets – effective, but only provide modest payback"
Server-Side components – provide significant payback but require extensive up-front design and an architectural foundation.
Infrastructure components – generic services for transactions, messaging, and database … require extensive design and complex programming
High-level patterns - identify components with high reuse potential
Packaged applications – only guaranteed reuse, … may not offer the exact functionality required

This article and the research behind it are concerned with the first three types of reuse.

Reusing code has several key implementation areas: application evolution, multiple implementations, standards, and new applications. The reuse of code from prior applications in new applications has received the most attention. However, just as important is the reuse of code (and the technology embedded therein) within the same application.

APPLICATION EVOLUTION

Charles Darwin stated that it was not the biggest, smartest, or fastest species that would survive, but the most adaptable. The same is true for application software. Applications must evolve even before they are completely developed, since the environment under which they operate (business, regulatory, social, political, etc.) changes during the time the software is designed and implemented. This is the traditional "requirements creep". Then after the application is successfully deployed, there is a constant need for change.

This paper appears in *Managing Information Technology in a Global Economy,* the proceedings of the Information Resources Management Association International Conference.

MULTIPLE IMPLEMENTATIONS

Another key need for reusability within the same application is for multiple implementations. The most common need for multiple implementations involves customizations, internationalization, and multiple platform support. Organizations whose software must be utilized globally may have a need to present an interface to customers in the native language and socially acceptable look and feel ("localization"). The multiple platform dimension of reuse today involves an architectural choice in languages and delivery platforms.

CORPORATE SOFTWARE DEVELOPMENT STANDARDS

Corporate software development standards concern both maintaining standards in all parts of an application and maintaining standards across all applications. "For a computer system to have lasting value it must exist compatibly with users and other systems in an ever-changing Information Technology (IT) world. [Brandon, 1999] As stated by Weinschenk and Yeo, "Interface designers, project managers, developers, and business units need a common set of look-and-feel guidelines to design and develop by." [Weinschenk, 1995] In the area of user interface standards alone, Appendix A of Weinschenk's book presents a list these standards; there are over three hundred items [Weinschenk, 1997]. Many companies today still rely on some type of printed "Standards Manuals."

ACHIEVING EFFECTIVE SOFTWARE REUSE

In most organizations, software reusability is a goal that is very elusive, as said by Bahrami "a most difficult promise to deliver on". [Bahrami, 1999] Radding stated: "Code reuse seems to make sense, but many companies find there is so much work involved, it's not worth the effort. …In reality, large scale software reuse is still more the exception than the rule" [Radding, 1998]. Bean in "Reuse 101" states; the current decreased "hype" surrounding code reuse is likely due to three basic problems [Bean, 1999]:

Reuse is an easily misunderstood concept

Identifying what can be reused is a confusing process

Implementing reuse is seldom simple or easy to understand

Grinzo also list several reasons and observations on the problem of reuse [Grinzo, 1998], other than for some "difficult to implement but easy to plug-in cases" such as GUI widgets: a "nightmare of limitations and bizarre incompatibilities", performance problems, "thorny psychological issues" involving programmers' personalities, market components that are buggy and difficult to use, fear of entrapment, component size, absurd licensing restrictions, or lack of source code availability.

Some organizations try to promote software reusability by simply publishing specifications on class libraries that have been built for other in house applications or that are available via third parties, some dictate some type of reuse, and other organizations give away some type of "bonus" for reusing the class libraries of others. [Bahrami, 1999]

But more often than not, these approaches typically do not result in much success.

> "It's becoming clear to some who work in this field that large-scale reuse of code represents a major undertaking" [Radding, 1998]. " An OO/reuse discipline entails more than creating and using class libraries. It requires *formalizing* the practice of reuse" [McClure, 1996].

Based upon both our literature research herein and experimental implementations, it was concluded that there were two key components to formalizing an effective software reuse practice both within an application development and for new applications. These components were:

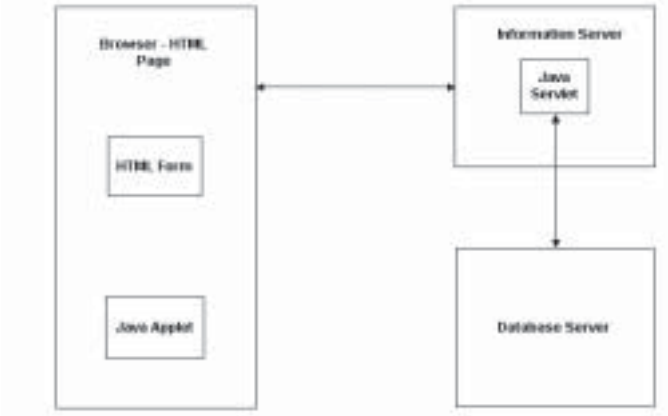
1. Defining a specific Information Technology Architecture within which applications would be developed and reuse would apply
2. Defining a very specific object oriented "Reuse Foundation" that would be implemented within the chosen IT architecture

IT ARCHITECTURE

"If you want reuse to succeed, you need to invest in the architecture first" [Radding, 1998]. "Without an architecture, organizations will not be able to build or even to buy consistently reusable components." In terms of IT architectures for business systems, there are historically several types as: Central Computer, File Services, Two or Three Tier Client Server, and Two or Three Tier Internet (Browser) based. Various transaction processing and database vendors have their own "slants" on these basic approaches, which may depend upon how business logic and the database are distributed.

It was decided to base our implementation research and development on the last of these categories as shown in Figure 2. Only vendor independent and "open" architectures would be used. The "multiple platform" dimension of reusability would be handled by using Java and Java generated HTML. Internet based applications are becoming the preferred way of delivering software based services within an organization (Intranets), to the worldwide customer base via browsers and "net appliances" (Internet), and between business (Extranets).

Figure 2



The presentation layer is represented by browser windows using HTML or Java Applets. The HTML is a static container for the Java Applet or is dynamically generated by a Java Servlet. The business logic layer is in the form of Java Servlets running on the information (Internet) server. The database, typically running on a separate server, is accessed via JDBC from the Servlets (or even from the Applets if a "type 4" pure JDBC driver was used).

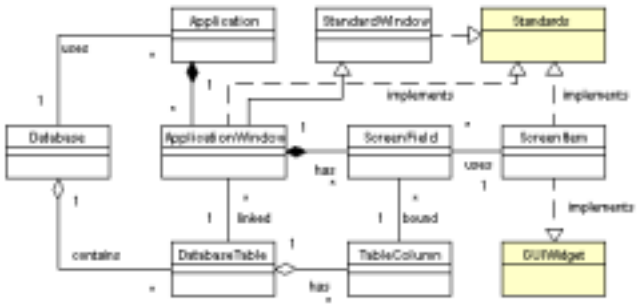
OBJECT ORIENTED REUSE FOUNDATION

As has been concluded by several authors, "A reuse effort demands a solid conceptual foundation" [Barrett, 1999]. The foundation used here is shown in Figure 3, and is called the "Object Oriented Reuse Foundation" [OORF]. It is based on the

key object oriented principles of inheritance and composition. By establishing this foundation, an organization can effectively begin to obtain significant reusability since programmers must inherit their class from one of the established classes and they must only compose their class of the established pre-built components.

In the design of Figure 3, an application is composed of a number of Application Windows. Each of these is derived from the Standard Window (or from another window which was derived from the Standard Window) and is associated with a table or view in that database. The Application Window implements the Standards interface. The Application Window is composed of screen fields, which use a specific screen item and are bound to a column of the database table/view. Each screen item implements the Standards and also implements the GUIWidget interface. The GUIWidget interface defines the functions that all screen items provide (such as: requestFocus, setText, getText, isValid, etc.). The screen items can be from the Java AWT, Java Swing, or third party class libraries as long as these class library sources have been extended to use the data in the Standards. The Standards interface defines all the standards used throughout the system including: fonts, colors, styles, sizes, initial states, icons, etc.

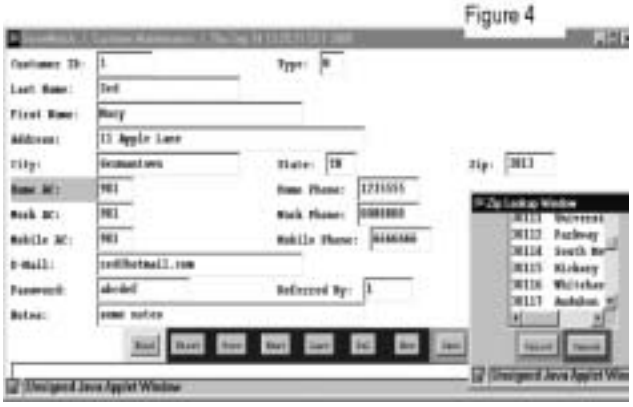
Figure 3



While Figure 3 shows the conceptual OORF, there would typically be an inheritance hierarchy of Standard Windows including forms, tables, etc. Screen Items would be a hierarchy also for the different types of these widgets such as textboxes, radio buttons, choice buttons, etc. Each application could also create an inheritance hierarchy of application windows.

Figure 4 shows a generated application window which provides navigation and update support for a selected database table including automatic lookup of defined foreign keys to maintain referential integrity. The reusability for this example was 95%, that is 95% of the lines of code were already in the OORF. For the applications implemented thusfar, all obtained reusability of over 90%.

Figure 4



REFERENCES

Bahrami, Ali. Object Oriented Systems Development, Irwin McGraw Hill, 1999

Barrett, Keith and Joseph Schmuller. "Building an Infrastructure of Real-World Reuse", Component Strategies, October 1999

Bean, James. "Reuse 101", Enterprise Development, October 1999

Bowen. Barry. "Software Reuse with Java Technology: Finding the Holy Grail", www.javasoft.com/features/1997/may/reuse.html

Brandon, Dan. "An Object Oriented Approach to User Interface Standards", Challenges of Information Technology in the 21st Century, Idea Group Publishing, 2000

Grinzo, Lou. "The Unbearable Lightness of Being Reusable", Dr. Dobbs Journal, September, 1998

Lim, Wayne C., Managing Software Reuse, Prentice Hall, 1998

McClure, Carma. "Experiences from the OO Playing Field", Extended Intelligence, 1996

Paulk, Mark, et. al. The Capability Maturity Model, Addison Wesley, 1995

Radding, Alan. "Hidden Cost of Code Reuse", Information Week, November 9, 1998

Reifer, Donald. Practical Software Reuse, Wiley Computer Publishing, 1997

Weinschenk, Susan and Sarah Yeo. Guidelines for Enterprise Wide GUI Design, John Wiley & Sons, 1995

Weinschenk, Susan, Pamela Jamar, and Sarah Yeo. GUI Design Essentials, John Wiley & Sons, 1997

Williamson, Miryam. "Software Reuse", CIO Magazine, May 1999