A Survey of Black-Box Modernization Approaches for Information Systems

Santiago Comella-Dorda
Software Engineering Insitute
scd@sei.cmu.edu

Kurt Wallnau
Software Engineering Institute
kcw@sei.cmu.edu

Robert C. Seacord
Software Engineering Institute
rcw@sei.cmu.edu

John Robert
Software Engineering Institute
jer@sei.cmu.edu

Abstract

Information systems are critical assets for modern enterprises and incorporate key knowledge acquired over the life of an organization. These systems must be updated continuously to reflect evolving business practices. Unfortunately, repeated modification has a cumulative effect on system complexity, and the rapid evolution of technology quickly renders existing technologies obsolete. Eventually, the existing information systems become too fragile to modify and too important to discard. For this reason, organizations must consider modernizing these legacy systems to remain viable. The commercial market provides a variety of solutions to this increasingly common problem of legacy system modernization. Understanding the strengths and weaknesses of each modernization technique is paramount to select the correct solution and the overall success of a modernization effort. This paper provides a survey of modernization techniques including screen scraping, database gateway, XML integration, CGI integration, object-oriented wrapping, and "componentization" of legacy systems. This general overview enables engineers performing legacy system modernization to preselect a subset of applicable modernization techniques for further evaluation.

Introduction

Information technology (IT) is revolutionizing commercial and government organizations by enabling efficiency improvements and creating opportunities for new business practices. Organizations depend on enterprise information systems (EISs) to codify these business practices and collect, process and analyze business data. No less an authority on the subject than Federal Reserve Chairman Alan Greenspan testified before the Joint Economic Committee (U.S. Congress on June 14, 1999) that current U.S. and global economic growth is rooted in advances in IT. Chairman Greenspan specifically credited IT, saying:

> *An economy that twenty years ago seemed to have seen its better days, is displaying a remarkable run of economic growth that appears to have its roots in ongoing advances in technology. Innovations in information technology—so-called IT—have begun to alter the manner in which we do business and create value, often in ways that were not readily foreseeable even five years ago.*[1]

In many ways, these information systems are to an enterprise what a brain is to the higher species—a complex, poorly understood mass upon which the organism relies for its very existence. John Salasin characterized enterprise information systems as large, heterogeneous, distributed, evolving, dynamic, long-lived, mission critical, systems of systems. EISs are large because the organizations they support are large

[1] Remarks by Chairman Alan Greenspan, *High-Tech Industry in the U.S. Economy,* testimony before the Joint Economic Committee, U.S. Congress. June 14, 1999.

1063-6773/00 $10.00 © 2000 IEEE

and complex, and because EISs evolve through accretion. Much of the code in these systems is likely to be redundant, often providing the same or similar capabilities in different subsystems that make up the overall EIS. EISs are heterogeneous for much the same reason. As features are added to an EIS, new technologies and components are selected and integrated.

The importance of EISs requires organizations to manage system evolution as business practices change and new information technologies that can provide competitive advantage become available. EIS evolution becomes more difficult with time as systems are repeatedly modified and become increasingly outdated. Managing the evolution of outdated systems requires periodically modernizing these legacy systems to support evolving business practices and to incorporate modern information technologies.

This document enumerates, describes and, to some extent, evaluates popular legacy system modernization techniques. In addition, we define system modernization and the role it plays in managing system evolution.

2. System Evolution

System evolution is a broad term that covers a continuum from adding a field in a database to completely re-implementing a system. These system evolution activities can be divided into three categories [Weiderman 97]: maintenance, modernization, and replacement. Figure 1 illustrates how different evolution activities are applied at different phases of the system life cycle. The dotted line represents growing business needs while the solid line represents the functionality provided by the information system. Repeated system maintenance supports the business needs sufficiently for a time, but as the system becomes increasingly outdated, maintenance falls behind the business needs. A modernization effort is then required that represents a greater effort, both in time and functionality, than the maintenance activity. Finally, when the old system can no longer be evolved, it must be replaced.

Determining the category of evolutionary activity that is most appropriate at different points in the life cycle is a daunting challenge. Should we continue maintaining the system or modernizing it? Should we completely replace the system? To make the correct decision, we need to assess the legacy system and analyze the implications of each action. Ransom et al describe an assessment technique for determining if a legacy system should be replaced, modernized or maintained [Ransom 98]. This document focuses on one phase in the life of a system: modernization. In particular, we concentrate on black-box modernization techniques because they provide a good way to leverage the existing investment in the legacy systems with a limited effort. To better understand the extent of modernization, however, first we briefly describe the other two phases in the life of a deployed system: maintenance and replacement.

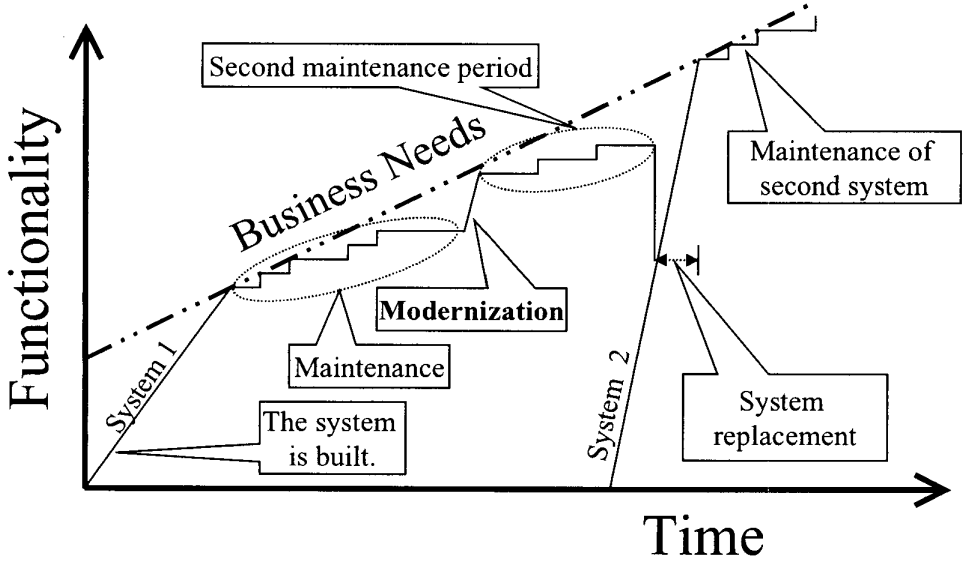

Figure 1: Information System Life Cycle

2.1. Maintenance

Maintenance is an incremental and iterative process in which small changes are made to the system. These changes are often bug corrections or small functional enhancements and should never involve major structural changes. Maintenance is required to support the evolution of any system, but it does have limitations:

1. Competitive advantage derived from adopting newtechnologies is seriously constrained. Netcentric computing or graphical user interfaces, for example, are not typically considered as a maintenance operation.

2. Maintenance costs for legacy systems increase with time. Finding needed expertise in out-of-date technologies becomes increasingly difficult and expensive.

3. Often, the compound impact of many small changes is greater than the sum of the individual changes due to the erosion of the system's conceptual integrity. Information systems tend to expand with time as efforts to remove unused code are seldom funded. Modifying a legacy system to adapt it to new business needs becomes increasingly difficult.

2.2. Replacement

Replacement (aka big bang approach or cold turkey) [Bisdal 97] is appropriate for legacy systems that can not keep pace with business needs and for which modernization is not possible or cost effective. Replacement is normally used with systems that are undocumented, outdated, or not extensible. However, replacement has risks that should be evaluated before selecting this technique:

1. Replacement is basically building a system from scratch and is very resource intensive. In addition, IT resources are typically fully allocated performing maintenance tasks and may not be familiar with new technologies that can be utilized on the new system.

2. Replacement requires extensive testing of the new system. Legacy systems are well tested and tuned, and encapsulate considerable business expertise. There is no guarantee that the new system will be as robust or functional as the old one (and as shown in Figure 1, it may cause a period of degraded system functionality with respect to business needs).

Replacement is a broadly studied problem with plenty of supporting literature. For a description of different replacement techniques see [Brodie 95]. Seng and Tsai illustrate in [Seng 99] a complete replacement process.

2.3. Modernization

Modernization involves more extensive changes than maintenance, but conserves a significant portion of the existing system. These changes often include system restructuring, important functional enhancements, or new software attributes. Modernization is used when a legacy system requires more pervasive changes than those possible during maintenance, but it still has business value that must be preserved.

System modernization can be distinguished by the level of system understanding required to support the modernization effort [Weiderman 97]. Modernization that requires knowledge of the internals of a legacy system is called white-box modernization, and modernization that just requires knowledge of the external interfaces of a legacy system is called black-box modernization.

2.3.1. White-Box Modernization. White-box modernization requires an initial reverse engineering process to gain an understanding of the internal system operation. Components of the system and their relationships are identified, and a representation of the system at a higher level of abstraction is produced [Chikofsky 90].

Program understanding is the principal form of reverse engineering used in white-box modernization. Program understanding involves modeling the domain, extracting information from the code using appropriate extraction mechanisms, and creating abstractions that help in the understanding of the underlying system structure.[2] Analyzing and understanding old code is a difficult task because with time, every system collapses under its own complexity [Phoenix Group]. Although some advances have been made in program understanding, it is still a risky and work-intensive task [von Mayrhauser 94, Haft 95].

After the code is analyzed and understood, white-box modernization often includes some system or code restructuring. Software restructuring can be defined as "the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics)." [Chikofsky 90] This transformation is typically used to augment some quality

[2] Tilley, Scott R. & Smith, Dennis B. *Perspectives on Legacy Systems Reengineering* (draft). Reengineering Center, Software Engineering Institute, Carnegie Mellon University. 1995.

attribute of the system like maintainability or performance. Program (or code) slicing is a particularly popular technique of software restructuring. A description of a semi-automatic restructuring technique to improve cohesion of legacy procedures can be found in [Lakhotia 98].

2.3.2. Black-Box Modernization. Black-box modernization involves examining the inputs and outputs of a legacy system within an operating context to gain an understanding of the system interfaces. Although acquiring an understanding of a system interface is not an easy task, it does not reach the degree of difficulty associated with white-box modernization.

Black-box modernization is often based on wrapping. Wrapping consists of surrounding the legacy system with a software layer that hides the unwanted complexity of the old system and exports a modern interface. Wrapping is used to remove mismatches between the interface exported by a software artifact and the interfaces required by current integration practices [Wallnau 97, Shaw 95]. Ideally, wrapping is a "black-box" reengineering task in that only the legacy interface is analyzed and the legacy system internals are ignored. Unfortunately, this solution is not always practical, and often requires an understanding of the software modules' internals using white-box techniques [Plakosh 99].

3. Modernization Techniques

Legacy systems may be modernized at the functional (logic), data, or user interface level. In this section, we present a collection of techniques for each of these modernization levels and discuss typical applications.

3.1. User Interface Modernization

The user interface (UI) is the most visible part of a system. Modernizing the UI improves usability and is greatly appreciated by final users.

A common technique for UI modernization is screen scraping [Carr 98]. Screen scraping consists of wrapping old, text-based interfaces with new graphical interfaces. The old interface is often a set of text screens running in a terminal. In contrast, the new interface can be a PC-based, graphical user interface (GUI), or even a hypertext markup language (HTML) light client running in a Web browser. This technique can be extended easily, enabling one new UI to wrap a number of legacy systems. The new graphical interface communicates with the old one using a specialized commercial tool.[3] These tools often generate the new screens automatically by mapping the old ones.

From the perspective of the legacy system, the new graphical interface is indistinguishable from an end user entering text in a screen. From the end user's point of view, the modernization has been successful as the new system now provides a modern, usable graphical interface. However, from the IT department's perspective, the new system is as inflexible and difficult to maintain as the legacy system. Screen scraping is basically a "makeover" for legacy systems. Derogatorily, this approach has been called "whipped cream on road kill." It can, however, be effective for stable systems where the principle objective is to improve usability.

Another interesting application of screen scraping is to generate application program interfaces (APIs) from legacy user interfaces. This technique was applied in a large defense program integrating an enterprise resource planning (ERP) with other systems. Screen scraping was used to extract data from the ERP, reversing the normal use of this wrapping technique because the ERP did not provide a callable application programming interface.

3.2. Data Modernization

Data wrapping enables accessing legacy data using a different interface or protocol than those for which the data was designed initially. Data wrapping improves connectivity and allows the integration of legacy data into modern infrastructures.

3.2.1. Database Gateway. A database gateway is a specific type of software gateway that translates between two or more data access protocols [Altman 99]. There are many vendor-specific protocols used to access databases, but fortunately there are a few that are de facto industry standards:

1. Open Database Connectivity (ODBC) is Microsoft's interface for accessing data in a heterogeneous environment of relational and non-relational database management systems. Based on the Call Level Interface specification of the SQL Access Group, ODBC provides an open, vendor-neutral way of accessing data stored in a variety of proprietar personal computer, minicomputer, and mainframe databases [Microsoft 95].

[3] For example, OC://WebConnect Enterprise Integration Server™ from Open-Connect Systems or QuickApp™ from Attachmate.

2. Java Database Connectivity (JDBC) is an industry standard defined by Sun for database-independent connectivity between Java applets/applications and a broad range of SQL databases. JDBC benefits from the "Write Once, Run Anywhere" characteristics of Java. The JDBC API defines Java classes that represent database connections, SQL statements, result sets, and database metadata [JDBC].

3. ODMG is the standard of the Object Data Management Group for persistent object storage. It builds upon existing database, object, and programming language standards (including the Object Management Group [OMG]) to simplify object storage and ensure application portability [Barry98].

A database gateway normally translates a vendor-specific access protocol into one of these standard protocols. This translation is useful because modern applications and development platforms typically support one or more of these standard protocols. Using a database gateway to access legacy data improves connectivity, enables remote access, and supports the integration of legacy data with modern systems.

Given that there are multiple "standard" protocols for accessing databases, the database gateway available for a specific legacy system and the protocol supported by the new system may not match. Figure 2, for example, shows a legacy system for which an ODBC gateway is available while the modern system requires a JDBC interface. One solution is a special gateway called a bridge that translates one standard protocol into another—in this case a JDBC-ODBC bridge.

3.2.2. XML Integration

The Extensible Markup Language (XML™) is a broadly adopted format for structured documents and data on the Web. XML is a simple and flexible text format derived from standard generalized markup language (SGML) (ISO 8879) and developed by the World Wide Web Consortium® (W3C).

XML is expanding from its origin in document processing and becoming a solution for data integration [Karpinski 98]. XML excels in inter-application data exchange because of its flexible and extensible method for describing data and its capability to communicate over the Internet using the standard HTTP protocol [WebMethods 99].

This flexibility makes XML a powerful mechanism for business-to-business (B2B) application integration. B2B integration is the automated exchange of information between systems from different organizations. B2B, for example, improves external processes such as supply chain integration or shipping/logistics tracking [WebMethods 99]. XML-based B2B is gaining momentum as XML vocabularies emerge in specific business domains such as finance. In addition, a growing number of commercial enterprise application solutions are embracing XML.

The keystone in the XML-based B2B architecture is the XML server (Figure 3). The XML server acts as the contact point between the corporate infrastructure and the rest of the world. The XML server communicates by various means with the internal infrastructures including ERP systems, databases, EDIs, etc. On the other hand, the server interoperates with external organization by exchanging XML messages. There is an active market of solutions for non-intrusive integration of legacy infrastructures into XML servers. In addition, most of the commercial XML servers support a wealth of communication protocols and this enables cost-effective integration with the most usual legacy applications.

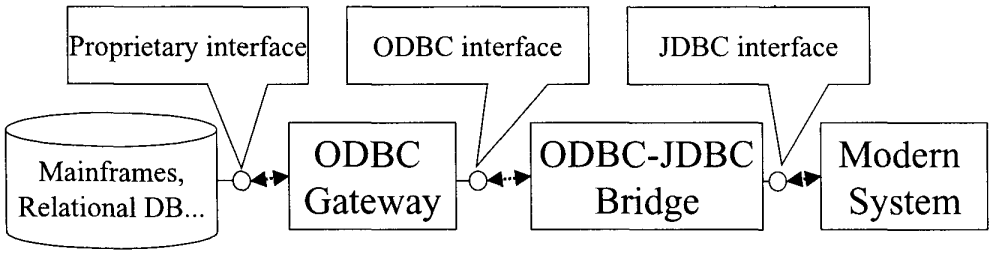

Figure 2: Gateways and Bridges

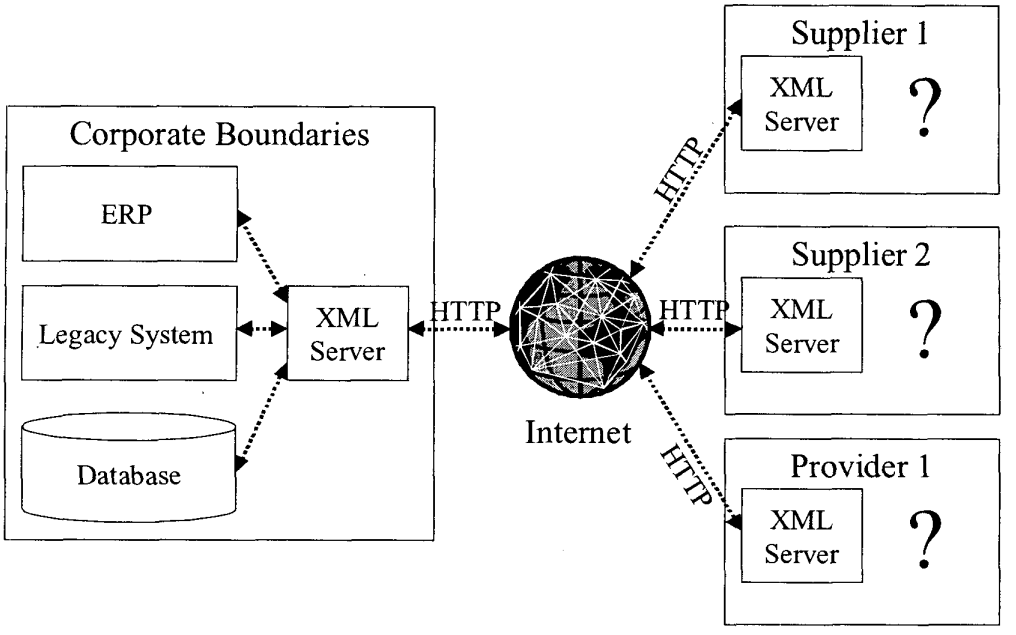

Figure 3: XML integration

3.3. Functional (Logic) Modernization

In contrast with data wrapping, functional (or logic) wrapping encapsulates not only the legacy data, but also the business logic embedded in the legacy system. Logic wrapping can be used, for instance, to leverage existing COBOL code implanted in Transaction Monitor (TM) procedures. The logic wrapping techniques presented here can provide access to legacy data, if required, in addition to legacy logic.

3.3.1. CGI Integration. The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. Legacy integration using the CGI [Shklar, Eichman 95] is often used to provide fast web access to existing assets including mainframes and transaction monitors. As in screen scraping, a new graphical user interface (in this case always HTML pages) is created, but instead of wrapping the old user interface, the new GUI communicates directly with the core business logic or data of the legacy system. CGI integration is more flexible than screen scraping because the new interface does not need to match the old user interface. However, it shares the advantages and disadvantages of screen scraping in that it is relatively easy to implement but does not fully address maintenance issues.

A typical CGI access configuration is shown in Figure 4. A Web server, powered with a CGI extension to access legacy systems, invokes some function in the legacy system and generates HTML pages to be served to remote browsers. Although not depicted in the figure, CGI is used to access legacy data in addition to the logic.

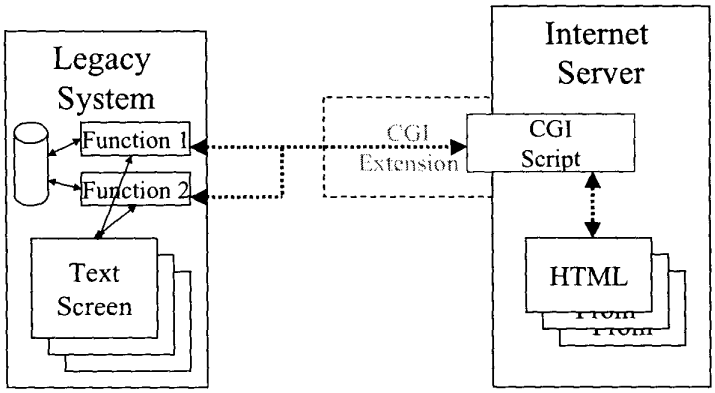

Figure 4: Legacy System Wrapping Using CGI Extensions

An example of using CGI to integrate legacy systems occurred in the National Software Data and Information Repository (NSDIR) when Perl scripts were used to access legacy data contained within the repository [Card 96].

3.3.2. Object-Oriented Wrapping. Objects have been used to implement complex software systems successfully. Object-oriented systems can be designed and implemented in a way that closely resembles the business processes they model [Phoenix Group]. Additionally, the use of abstraction, encapsulation, inheritance, and other object orientation (OO) techniques make object-oriented systems easier to understand.

To support object distribution, we need a more powerful means of communication than that provided by normal inter-object communication mechanisms. Distributed object technology (DOT) is the combination of distributed technology with OO [Wallnau 97]. In effect, DOT extends object technology to the net-centric information systems of modern enterprises by using *object middleware*. The most prevalent object middleware is the Common Object Request Broker Architecture (CORBA) from OMG, with its platform-neutral object specification language, robust remote method calls, interoperable protocols, and rich set of services.

The conceptual model of object-oriented wrapping is deceptively simple: individual applications are represented as objects; common services are represented as objects; and business data is represented as objects. In reality, object-oriented wrapping is far from simple and involves several tasks including code analysis, decomposition, and abstraction of the OO model. The project ERCOLE (Encapsulation, Reengineering, and Coexistence of Object with Legacy) describes an exemplifying process to wrap legacy applications with OO systems [De Lucia 97]. Of the multiple technical difficulties involved in wrapping a legacy system, two are of special relevance: the definition of appropriate object-level interfaces and the need for integrated infrastructure services.

Translating the monolithic and plain semantics of the often procedural legacy system to the richly hierarchic and structured semantics of an object-oriented system can be a difficult task. A good knowledge of the domain can greatly help in the translation. For example, Stets describes an experience in which the Win32 API is translated into objects [Stets 99]. To create meaningful objects, domain-specific knowledge of the structure of an operating system is used. Unfortunately, developers wrapping a system rarely possesses such deep domain knowledge. Some techniques have been developed to perform the legacy to OO mapping more automatically. For example, in one such method, every coarse-grained persistent item is mapped into an object, and services are assigned to objects with an algorithm that minimizes coupling [Cimitile 97]. Although these and other techniques are useful in extracting objects from legacy systems, the mapping problem is far from being solved.

The second challenge of translating an OO system using DOT is integrating infrastructure services in the OO system. Almost any DOT middleware provides some set of services such as security, transactions or persistence. However, it is often the case that to get the expected level of services the developer must integrate two or more of these middleware solutions. This integration is at least problematic due to unexpected interactions and incompatibilities between theoretically compatible products [Seacord 99]. To address this need for integrated services, the industry has developed what is commonly known as application servers (aka server-side component frameworks). An application server is a product that integrates a set of services and defines a component model. Components developed conforming to this model

are able to leverage all the services provided by the application server.

The OMG is following this approach in the CORBA 3 specification, which defines a component framework (known in CORBA 3 jargon as component container) and addresses a range of infrastructure issues as well.

3.3.3. Component Wrapping. Component wrapping is very similar to OO wrapping, but components, in contrast with objects, must conform to a component model. This constraint enables the component framework to provide the component with quality services. Because component-based modernization is the newest and probably the least known of the approaches illustrated in this survey, this section discusses not only an overview but some implementation details as well.

The component model market consisted initially of numerous proprietary and mutually incompatible solutions. Fortunately, this situation is improving with the emergence of a small set of standard enterprise component models and products. Of these standards, three are of particular consequence:

- Distributed interNet Architecture™ (DNA) is the Microsoft solution and a de facto standard because of Microsoft's weight in the industry.
- The CORBA 3 Component Model is the OMG approach for the enterprise that promises an enterprise component model for CORBA.
- Enterprise JavaBeans (EJB) is the Sun Microsystems solution for Java server-side computing [Johnson 98].

According to press releases from the OMG, it is likely that the CORBA 3 Component Model and EJB will merge into a single standard leaving only two dominant players. The following discussion uses EJB to illustrate component wrapping, but does not necessarily mean that EJB is superior to other competing models or that these solutions cannot be implemented with other products.

Components in EJB are known as Enterprise JavaBeans (we will also refer to them as beans). Each bean encapsulates a piece of business logic. Enterprise JavaBeans are deployed within an application server, or EJB server, that provides the runtime environment for the bean and manages common services such as security, transactions, state management, resource pooling, distributed naming, automatic persistence, and remote invocation. This allows the EJB developer to focus on the business problem to be solved. An Enterprise JavaBean is, in essence, a transactional and secure remote method invocation (RMI) or CORBA object, some of whose runtime properties are specified at deployment using special files called "deployment descriptors."

The first step in wrapping a legacy system using EJB is to separate the interface of the legacy system into modules consisting of logical units—shown in Figure 5 as Function 1 and Function 2. The degree of difficulty in dividing the legacy system into discrete functions will vary depending on the degree to which the separation was defined in the legacy system interfaces and whether new interfaces must be built. Although a black-box approach is preferable, poor documentation and a cryptic legacy system interface may make it necessary to peer into the black-box to better understand the legacy system [Plakosh 99].

The next step in wrapping the legacy business logic is to build a *single point of contact* to the legacy system. It is a good idea to centralize all the communication knowledge in a single software artifact. The communication method used by this artifact depends on the particular situation. Options for communication between the single point of contact and the legacy system include RMI over IIOP, sockets, or even Message Oriented Middleware (MOM), which has the advantage of uncoupling the EJB server from the legacy system and allowing for asynchronous communication. This single point of contact can be implemented as a bean (called *adapter*) or as a *service broker*—a software component placed external to the EJB server. Placing the point of contact inside or outside the server depends mainly on the chosen communication method and some security restrictions. For example, if you need to create a new thread or listen to a socket, the single point of contact must be outside of the application server because the EJB specification prevents Enterprise JavaBeans from being multithreaded or listening to sockets. The final step in wrapping the legacy business logic is to implement a wrapper bean for each module in the legacy system. In Figure 5, this wrapper is shown as Bean 2. These beans forward requests to the legacy system using the single contact point, in a manner similar to object wrapping.

There are several advantages to the component approach for wrapping legacy business logic. First, with relatively limited effort, the advantages of component-based systems are supported. We can, for example, build new Enterprise Java Beans that use the wrapper beans in unanticipated ways, greatly improving system flexibility. Second, wrapper beans are bona fide Enterprise Beans and can be integrated fully with all the management facilities and services included with the application server. Lastly, wrapping legacy business logic provides a roadmap to substitute the old system incrementally. After wrapping the functionality of the legacy system, we can re-implement wrapper beans one at a time (Bean 1 in Figure 5), without having to go through a "big-bang" replacement of the system. This is possible because the system and the clients would not notice any disruption as

the re-implemented bean maintains the same interfaces provided by the wrapper bean. In time, it is possible to replace the old system completely.

Wrapping legacy business logic with EJB is not without risk. The EJB specification is porous and portability problems can arise between different vendor's application servers [Comella-Dorda 99]. In addition, the Java programming language is considered by some to be unsuitable for critical applications.

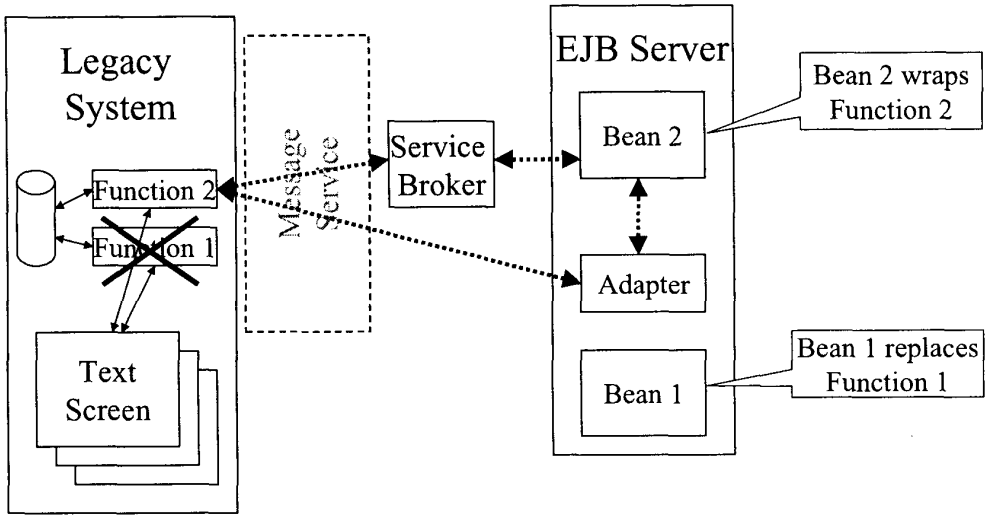

Figure 5: Wrapping Legacy Business Logic Using EJB

4. Summary

The modernization of legacy systems is a critical issue that IT departments must face in modern enterprises. Although new technologies such as CORBA and EJB promise to support modernization, it is still a complex task requiring careful consideration and advanced planning. There are different approaches to the modernization of legacy assets including reengineering (white-box) and wrapping (black-box). Before starting any legacy modernization effort, every possible option should be explored and evaluated. This evaluation should cover the technical considerations and consider business and strategic factors outlined in [Weiderman 97b, Bergey 97] to ensure long-term success.

We have presented several techniques to support legacy system modernization. It would be naive to affirm any of these techniques as superior to the others. Each presented technique has strengths, weaknesses, and tradeoffs between cost, flexibility, and other variables. Table 1 summarizes the discussions of each presented technique.

Table 1: Comparison of Modernization Techniques

	Artifact Modernized	Target	Strengths	Weaknesses
Screen Scraping	Text-based user interface	Graphical or web-based user interface	• Cost • Time to market • Internet support	• Flexibility • Limited impact on maintainability
Database Gateway	Proprietary access protocol	Standard access protocol	• Cost • Tool support	• Limited impact on maintainability

XML Integration	Proprietary access protocol	XML server	• Flexibility • Tool support (future) • B2B	• Tool support (present) • Evolving technology
CGI Integration	Mainframe Data or TM services	HTML pages	• Cost • Internet support	• Flexibility • Applicability
OO Wrapping	Any Enterprise Resource	OO Model	• Flexibility • Easier understanding	• Cost
Component wrapping	Any Enterprise Resource	Component Model	• Flexibility • Integrated services • Incremen. replacement	• Cost

References

[Altman 99] Altman, R.; Natis, Y.; Hill, J.; Klein, J.; Lheureux, B.; Pezzini, M.; Schulte, R.; & Varma S. Middleware: The Glue for Modern Applications. Gartner Group, Strategic Analysis Report; 26 July 1999.

[Barry 98] Barry, Doug. "ODMG 2.0: A Standard for Object Storage." Component Strategies. July 1998.

[Bergey 97] Bergey, John; Northrop, Linda; & Smith, Dennis. Enterprise Framework for the Disciplined Evolution of Legacy Systems (CMU/SEI-97-TR-007). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/97.reports/97tr007/97tr007abstract.html> 1997.

[Bisdal 97] Bisdal, Jesus; Lawless, Deirdre; Wu, Bing; Grimson, Jane; Wade, Vincent; Richardson, Ray; & O'Sullivan, D. "An Overview of Legacy Information System Migration," Proceedings of the 4th Asian-Pacific Software Engineering and International Computer Science Conference (APSEC 97, ICSC 97), 1997.

[Brodie 95] Brodie, M. & Stonebraker, M. Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach. Morgan Kaufmann Publishers, 1995.

[Card 96] Card, D.N.; Hissam, S. A; & Rosemeier R.T. "National Software Data and Information Repository." CrossTalk 9, 2. Available WWW <URL: http://www.stsc.hill.af.mil/CrossTalk/1996/feb/national.html> (February 1996).

[Carr 98] Carr, David F. "Web-Enabling Legacy Data When Resources Are Tight." Internet World (August 10 1998).

[Chikofsky 90] Chikofsky, Elliot J. & Cross II, J.H. "Reverse Engineer and Design Recovery: A Taxonomy." IEEE Software, 7 (January 1990): 13-17.

[Cimitile 97] Cimitile, A.; De Lucia, A.; Di Lucca, A.; & Fasolino, A.R., "Identifying Objects in Legacy Systems," Proceedings of the 5th Workshop on Program Comprehension (WPC97), 1997.

[Comella-Dorda 99] Comella-Dorda, Santiago; Robert, John; & Seacord, Robert. "Theory and Practice of Enterprise JavaBean Portability," Proceedings International Society for Computers and Their Applications (ISCA) 1st International Conference on Information Reuse and Integration (IRI-99). Atlanta, Georgia, 1999.

[De Lucia 97] De Lucia, A.; Di Lucca, G.A.; Fasolino, A.R.; Guerra, P.; & Petruzzelli, S. "Migrating Legacy Systems towards Object-Oriented Platforms," International Conference of Software Maintenance (ICSM97), 1997.

[Eichman 95] Eichmann, David. Application Architectures for Web-Based Data Access. Available WWW: <URL: http://www.cs.rutgers.edu/~shklar/www4/eichmann.html>.

[Haft 95] Haft, T. M. & Vessey, I. "The Relevance of Application Domain Knowledge: The Case of Computer Program Comprehension." Information Systems Research, 6. (1995): 286-299.

[JDBC] Introduction to JDBC. Available WWW <URL: http://shrike.depaul.edu/~pgrage/dist_sys/ds520_p1.html>.

[Johnson 98] Johnson, Mark. "A Beginner's Guide to Enterprise JavaBeans." Java World 3, 10. Available WWW <URL: http://www.javaworld.com/javaworld/jw-10-1998/jw-10-beans.html> (October 1998).

[Karpinski 98] Karpinski, Richard. "Databases, Tools Push XML Into Enterprise." Internet Week Online. Available WWW

<URL: http://www.internetwk.com/news1198/news111698-3.htm> (November 1998).

[Lakhotia 98] Lakhotia, Arun & Deprez, Jean-Christophe. "Restructuring Functions with Low Cohesion," Proceedings of the 6th Working Conference of Electrical and Electronics Engineers, 1998.

[Microsoft 95] Microsoft. "Microsoft Windows Operating Systems and Services Architecture, Chapter 9: Open Database Connectivity (ODBC) 2.0 Fundamentals", Microsoft Corporation. 1995.

[Phoenix Group] Phoenix Group. Legacy Systems Wrapping with Objects. Available WWW <URL: http://www.phxgrp.com/jodewp.htm>.

[Plakosh 99] Plakosh, Daniel; Hissam, Scott; & Wallnau, Kurt. Into the Black Box: A Case Study in Obtaining Visibility into Commercial Software (CMU/SEI-99-TN-010). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University. Available WWW: <URL: http://www.sei.cmu.edu/publications/documents/99.reports/99tn010/99tn010abstract.html> (1999).

[Ransom 98] Ransom, J.; Sommerville, I.; & Warren, I. "A Method for Assessing Legacy Systems for Evolution," Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering (CSMR98), 1998.

[Seacord 99] Seacord, Robert C.; Wallnau, Kurt; Robert, John; Comella-Dorda, Santiago; & Hissam, Scott A. "Custom vs. Off-the-Shelf Architecture," Proceedings of 3rd International Enterprise Distributed Object Computing Conference, University of Mannheim, Germany, September 27-30, 1999.

[Seng 99]Seng, Jia-Lang & Tsai, Wayne. "A Structure Transformation Approach for Legacy Information Systems- A Cash Receipts/Reimbursement Example," Proceedings on the 32nd Hawaii International Conference on System Sciences, 1999.

[Shaw 95] Shaw, Mary. "Architecture Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging," Proceedings IEEE Symposium on Software Reusability. April 1995.

[Shklar] Shklar, Leon. Web Access to Legacy Data. Available WWW: <URL: http://athos.rutgers.edu/~shklar/web-legacy/summary.html>.

[Stets 99] Stets, Robert J.; Hunt, Galen C.; Scott, Michael L. "Component-Based APIs for Versioning and Distributed Applications," IEEE Computer, 54-61 (July 1999).

[Sun 99] Sun. J2EE™ Connector Architecture (JSR-000016). Available WWW: <URL: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_016_connect.html> (1999).

[von Mayrhauser 94] von Mayrhauser, A. & Vans, A.M. "Comprehension Processes During Large Scale Maintenance." Proceedings of the International Conference of Software Engineering ICSE. Sorrento, Italy, p. 39-48. May 16, 1994.

[Wallnau 97] Wallnau, Kurt; Morris, Edwin; Feiler, Peter; Earl, Anthony; Litvak, Emile. Engineering Component-Based Systems with Distributed Object Technology Lecture Notes in Computer Science. International Conference WWCA'97 Tsukuba, Japan. March 1997.

[WebMethods 99] WebMethods. B2B Integration: The Drive to Gain and Maintain Competitive Advantage. Available WWW: <URL: http://www.webmethods.com/products/whitepapers/b2b_wpB2Bintegration.html> (July 1999).

[Weiderman 97a] Weiderman, N; Northrop, L.; Smith, D.; Tilley, S.; & Wallnau, K. Implications of Distributed Object Technology for Reengineering (CMU/SEI-97-TR-005 ADA326945). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/97.reports/97tr005/97tr005abstract.html> (1997).

[Weiderman 97b] Weiderman, Nelson H.; Bergey, John K.; Smith, Dennis B.; & Tilley, Scott R. Approaches to Legacy System Evolution (CMU/SEI-97-TR-014). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University. Available WWW <URL:http://www.sei.cmu.edu/publications/documents/97.reports/97tr014/97tr014abstract.html> (1997).