Migration of Legacy Software to Service Oriented Architecture

Edward Stehle, Brian Piles, Jonathan Max-Sohmer, Kevin Lynch
Department of Computer Science
Drexel University
Philadelphia, PA 19104

Abstract

Legacy software is in use at a great number of organizations. Although organizations commonly get a great deal of value from their legacy systems eventually most legacy systems need to be modernized. In order to bring software systems up to date with current technologies, businesses have two choices. They can create new software to take the place of existing systems, or they can migrate current systems into a framework which utilizes current technologies. There are many advantages to migrating existing systems. Migrating existing systems can save considerable time and money compared to writing new systems and can help avoid creation of new bugs. In this paper we will examine methods of migrating legacy software to a current technology, specifically service oriented architectures.

1. Introduction

Computing systems have been a part of businesses and other large organizations for decades. Consequently much of the software in use today is decades old. Older systems, commonly referred to as legacy software, have great value to the organizations that use them, but create maintenance issues as they age. Eventually the drawbacks of legacy systems out weigh the value they bring to organizations that use them, and software resources must be modernized[5].

One of the most popular target architectures for modernization of software is service oriented architectures. SOAs are intended to handle modern net-centric distributed software needs, and are language and platform neutral. These features make SOA an ideal choice for modernization[3]. In this paper will focus specifically on modernizing legacy systems by moving them to SOA.

The modernization of software systems can be carried out in several ways. In this paper we will look at three common methods for modernization. First we will look at redevelopment, or complete re-write of the system from scratch. Second we will discuss wrapping, a method where the legacy system is wrapped in an interface which allows it to integrate with new technologies. Finally we will look at migration. Although all of these methods can be thought of as methods of migration, we will use the term in the same manner as Bisabal et al.[5] who use migration to refer to methods that use elements of both redevelopment and wrapping.

2. Legacy Software

Legacy software is commonly defined as any software which uses out-of-date technology. Most of the software used in businesses today falls into the category of legacy software.[1] As the rate of change in software technology increases, the rate at which systems move into the category of legacy software increases. According to Sneed the rate of change in technology is such that, 'Any user organization which has been using information technology for more than five years has a legacy software problem.'[1]

There is usually a good reason why a piece of software has persisted in an organization long enough to be considered legacy software. Software that works well, supports an organizations needs, and provides support in a dependable manner, tends to stick around. Large software systems represent a significant financial investment. Years of use lead to well tested systems. Legacy systems may be critical to the operation of an organization. A company may require close to %100 up time from their software. As Bisabal et al. state 'Legacy information systems are typically the backbone of an organizations information flow and the main vehicle for consolidating business information'[5]. Managers are reluctant to incur the cost and risk involved in replacing expensive dependable systems that are critical to company operations, but the downside of legacy systems can eventually out weigh the advantages.

Legacy software can be difficult to maintain due to a difficulty understanding the system and dependence on out-dated technologies. The authors of the software may no longer be available and the documentation may be insufficient or nonexistent. The lack of comprehension this creates causes difficulty in modifying and extending software. It may be difficult to find engineers who are proficient in legacy software languages and technologies. Even if the code can be understood, legacy software may depend on out-of-date hardware. Old hardware can be difficult to maintain and replacement parts may not be available. The performance of older hardware may no longer be acceptable.

3. Redevelopment

Redevelopment of legacy applications involves rewriting legacy applications ground up. A general approach for redevelopment is proposed by Zhang et al. [4] First the legacy code is analyzed to gain an understanding of its functionality. This analysis is used to identify business processes carried out by the legacy system. These business processes are then implemented as services.

There are several advantages to redevelopment. The new software will be designed from the ground up to be used in a SOA. Because the system is written in a manner that supports SOA it is easier to maintain and extend within SOA. Redeveloped software does not have the extra layers added by wrapping, and therefore does not suffer for the performance loss caused by extra layers.[1]

Although redevelopment produces software with desirable qualities, its cost in time and money can be prohibitive. Of the methods of modernization discussed in this paper redevelopment does the least to leverage the organization's investment in the legacy system. The legacy system is used only as a means to understand what needs to be implemented in the new system, none of the original code is re-implemented. Extracting the needed information from the legacy system is a

slow and difficult process. Implementing the new system is also a time consuming process. Long, difficult processes lead not only to a long wait for delivery, but also produce lots of man hours and great expense. Of the methods we discuss in this paper redevelopment is the most expensive and requires the most time. According to Bisbal et al. 'In reality, the risk of failure is usually too great for organizations to seriously contemplate a redevelopment approach.'[5]

4. Wrapping

Wrapping is another method to implement a SOA from legacy code; the main distinction in this process is the gluing of useful legacy code with wrapper code to incorporate it into a SOA system.

There are several ways to accomplish this, but the methodology proposed by Sneed[1] is effective in decoupling code from the original system. The relevant legacy code must first be determined. This can typically be achieved through automated reverse engineering tools; the code segments that perform a desired service or data modification are observed through clustering tools and data flow charts. These code segments are extracted, and a new component is built using these segments. The interface of the new component is then formed using the required data objects. This will finalize the decoupling of the function from its original legacy application and interface.

The newly created component is then given a WSDL interface, and a SOAP framework is used to build the component into an XML schema. Other tools can also be used to perform message parsing and forming return messages. This specific wrapping stage can also be done by automated software.

Finally a proxy is made to link the new services into the SOA architecture. The proxy is concerned with checking parameters and generating the required WSDL. This allows the wrapped legacy code to be incorporated into the SOA model.

This approach for implementing a SOA from legacy code is effective in some cases. The entire system does not have to be reengineered, and time can be saved because much of the wrapping can be done automatically. The complicated part of the process deals with extracting and decoupling applicable code and creating an appropriate interface to be wrapped. The rest of the process is simple and automated.

The process does have some negative issues and problems. Wrapping legacy components that are not specifically designed for SOA does incur a penalty in both the design and runtime of the system. Future modification and maintenance of the components may eventually become an issue since their design is not of SOA origin. Both the wrappers and program interfaces may need to be adjusted as the components evolve.

Additionally, introducing wrappers and proxies to perform parsing, messaging, and WSDL generation creates an overhead in the architecture. Although, the overhead for a component may not be substantial, having several interacting components that have been implemented in this fashion could cause noticeable system performance issues.

Wrapping is a useful approach when the issue meets several conditions. If the code is too expansive to completely redevelop, the desired processes can be decoupled from the system, and a quick, cost effective solution is required. Wrapping is not optimal, but it allows a traditional system to easily gain many of the benefits of service oriented architecture.

5. Migration

Migration in a loose sense is the purpose of all of these techniques; however, specifically it is a variation of the wrapping methodology. The migration process is similar, but it deals with incorporating different amounts of wrapping and application redevelopment.

Legacy code is found, decoupled, and extracted by similar methods. Following this, user interfaces are reengineered to adhere to a SOA structure. The entire component is not simply wrapped but, if possible, the interfaces are recreated in this manner. Some wrapping at the object level may still be necessary, for example, to interpret a web request that a language may not support.

A case study by Aversano[3] migrated a COBOL system to a web-based service oriented architecture using this methodology. The server interface of the old system was the main piece that was wrapped for integration into the new system. Other components user interfaces were reengineered for integration. This approach minimizes the amount of wrapping that is required to promote the design of the architecture and long-term migration of the system.

This technique is a mixture of the previous two; however, the level of granularity of redevelopment and wrapping is adjusted to produce a solution that is balanced in terms of efficiency, design, and cost. This methodology uses the concepts and best practices of the others to allow legacy systems to be migrated to SOA.

6. Conclusion

Migration will be an ever-existent part of the business process. Frameworks are increasing in size and complexity, and new functionality is being implemented on existing systems; therefore, migration is an important factor to address for the overall improvement and efficiency of a business.

The attractiveness of SOA is ever-increasing in computing and web services. Companies are seeking solutions to move existing code into service structures to improve accessibility and functionality. However, the transferring of these legacy systems must be analyzed for efficiency in time, cost, design, and functionality; efficient implementations of a system that follows good design principles must be created economically.

Many of the methods that exist to perform this migration excel in achieving one of these attributes: migrating costs or good design, for example. However, a combination of these methods with a long-term goal of complete migration is an optimal approach in a typical situation. Redevelopment can be done on very limited segments, and wrapping can be performed at a very fine granularity to quickly construct efficiently designed code. Although this is not the best system in terms of performance or design, it satisfies the constraints of the business while producing a solution that is as close to optimal as possible.

References

[1] Harry M. Sneed, “Integrating legacy Software into a Service oriented Architecture” csmr, pp. 3-14, Conference on Software Maintenance and Reengineering (CSMR’06), 2006

[2] Kishore Channabasavaiah, Kerrie Holley , Edward Tuggle, Jr. “Migrating to a service-oriented architecture”, IBM DeveloperWorks, http://www.ibm.com/developerworks/webservices/library/ws-migratesoa/

[3] L Aversano, G Canfora, A Cimitile, A De Lucia, “Migrating legacy systems to the Web: an experience report”, Fifth European Conference on Software Maintenance and Reengineering, 2001. Pages:148 - 157

[4] Z Zhang, H Yang, “Incubating services in legacy systems for architectural migration”, 11th Asia-Pacific Software Engineering Conference, 2004. Pages:196 - 203 t

[5] Bisbal, J.; Lawless, D.; Bing Wu; Grimson, J., “Legacy information systems: issues and directions”, Software, IEEE , vol.16, no.5, pp.103-111, Sep/Oct 1999