Legacy System Evolution towards Service-Oriented Architecture

Asil A. Almonaies, James R. Cordy, and Thomas R. Dean

School of Computing, Queens University
Kingston, Ontario, Canada
{almonaies,cordy,dean}@cs.queensu.ca

Abstract. Although Service-Oriented Architecture (SOA) has become popular in recent years, the majority of legacy systems are still not SOA enabled. The increase in the amount of information that companies must handle has resulted in a considerable increase in the complexity of the legacy systems that store this information. While moving to a service-oriented architecture platform can help in handling this increase, at the same time it is important to preserve the investment of many years of tuning and debugging of the legacy assets as much as possible. Several techniques exist for modernizing legacy systems towards service-oriented architecture. In this paper we present a survey of the various approaches to moving legacy systems to the SOA environment. We discuss the various approaches and methods and highlight their strengths and weaknesses, with an eye to assisting the decision process when undertaking a new modernization project.

Key words: SOA, legacy systems, wrapping, redevelopment, migration

1 Introduction

Service-oriented architecture (SOA) can be viewed as an architectural construct for flexible connection of separate components in response to changes in business. SOA focuses on the exchange of information among major software components and on the reusability of the components by separating the interface from the internal implementation. There are several features of SOA that make legacy system modernization appealing in today's world, including loose coupling, abstraction of underlying logic, agility, flexibility, reusability, autonomy, statelessness, discoverability and reduced costs. The primary purpose for the adoption of SOA is to improve business communication so that the goals of the enterprise can be more readily realized.

Chatarji [1] provides a summary of the business advantages of migration to SOA. Short-term benefits include improving reliability, reducing hardware costs, leveraging existing development skills and moving to a standards-based server and applications. An important advantage is providing a data bridge between incompatible technologies. The long-term benefits are reduced management costs and the collection of a unified information taxonomy.

In this paper, we provide a critical review of the existing literature in the area of legacy system modernization strategies to service-oriented architecture.

We divide the approaches into four categories: *replacement*, which rewrites the existing systems' applications or replaces it entirely with an off the shelf product, *redevelopment* or *reengineering*, where reverse engineering and reengineering approaches are used to add SOA functionality to legacy systems; *wrapping*, which provides a new interface to existing components to make them easily accessible as services to other software components; and *migration*, which moves the legacy system to the more flexible SOA environment while preserving the original system's data and functionality.

2 Comparison Criteria

We compare the approaches with respect to the following eight criteria:

- *Modernization Strategy.* The strategy of the proposed approach: one of replacement, redevelopment, wrapping and migration. The rest of the paper is structured based on this criterion.
- *Legacy System Type.* The kind of system to which the technique applies.
- *Degree of Complexity.* Time/cost complexity of the method (or NA, if not reported).
- *Analysis Depth.* The strategy used to analyze the legacy system to understand its concepts and locate the important functions to be exposed as part of SOA architecture. The analysis could be shallow or deep depending on the strategy used. Minimal dependency on the existing legacy system components in achieving SOA architecture can provide more flexibility.
- *Process Adaptability.* How well the process adapts to the legacy system to minimize the extent of the required modifications.
- *Tool Support.* To what degree is the process automated, and if a tool is proposed or implemented.
- *Degree of Coverage.* Does the proposed approach present a complete strategy for moving to SOA, or only a specific part of the modernization.
- *Validation Maturity.* Has the proposed approach been applied and validated. We classify the proposed approach as an idea, a method demonstrated by a case study, or a commercially proven technique.

3 Replacement of Legacy Systems

Although replacement is not one of the strategies advocated by the surveyed papers, it may make sense to retire the application and replace it with an off-the-shelf package or a complete rewrite of the legacy system from scratch. Two possible reasons are if the business rules in the application are well understood in the organization, and the legacy system involves obsolete or difficult to maintain technologies. An organization may choose the replacement strategy if wrapping, redevelopment, and migration will impose costs that cannot be justified. Rewriting the application from scratch is expensive, risky and time consuming, but has the advantage that it delivers a customized solution that can be built exactly to meet the organization's needs.

Commercial off-the-shelf (COTS) systems are ready-made, commercially available software products. Replacing the application with a COTS component,

while less risky and time consuming than rewriting, can also be expensive since future modifications may be difficult and expensive to perform. There are some cases where a COTS solution may not be a good option: important business are embedded in the legacy system, modification of the COTS package is expensive, or the loss of control of the software code base by the organization.

Replacement can take place either by using a "big-bang" strategy or incrementally. If the legacy system has a well defined structure, then it makes most sense to replace it incrementally. Comella-Dorda et al. [2] identify two significant risks of the replacement strategy: the maintenance of the new system, which will not be as familiar as the old system; and the lack of a guarantee that the new system will be as functional as the original. Given these risks and the reuse goals of service-oriented systems, in most cases we consider the replacement strategy for legacy code to be the least desirable solution for migration to SOA.

4 Wrapping Strategies

Wrapping provides a new SOA interface (e.g. WSDL) to a legacy component, making it easily accessible by other software components. It is a black-box modernization technique, since it concentrates on the interface of the legacy system, hiding the complexity of its internals. Wrapping is used when the legacy code is too expensive to re-write, is relatively small, can be reused, and a fast, cost-effective solution is needed. Wrapping gives legacy systems the benefits of service oriented architecture in a quick and a simple manner. If the legacy system has a high business value and good quality code, wrapping can be a good option.

The main problem is that this strategy does not change the fundamental characteristics of the legacy applications that are being integrated. Wrapping will not solve problems already present, such as problems in maintenance and upgrading. In many cases, studying the internals of the legacy system is important and white-box modernization tools are required.

4.1 Overview of wrapping techniques

Sneed [3, 4] discusses a tool-supported method for maintaining legacy code within an SOA environment. Legacy code is wrapped in an XML shell which allows individual functions in the programs to be offered as web services. The code segments that perform a desired service or data modification are identified using clustering tools and data flow, are extracted, and a new component is built using them. The new component is given a WSDL interface, and a SOAP framework is used to package the component. Finally, a proxy is made to link the new services into the SOA architecture. The technique has been illustrated by wrapping a COBOL calendar function extracted from the legacy software of a Swiss bank, and the approach has been applied successfully to the SOA integration of both COBOL and C++ programs. It is most suitable for smaller programs since identifying and exposing business functions can be time consuming.

Canfora et al. [5] propose a method to make the interactive functionalities of legacy systems accessible as Web Services by wrapping them in an SOA interface. The method provides the legacy system with a request/response interface, where

a client invokes a service using a request message and the provider responds with the required results. The wrapper interprets a Finite State Automaton which models the interaction between the client and the legacy system. They generate a wrapper for the email client Pine, which provides an SOA interface to the get message functionality. In more recent work [6], this wrapping technique is used as a part of a complete migration process consisting of the selection of the desired services, wrapping of the selected use cases and deployment and validation of the wrapped use cases. The main problem with the technique is that most of the work is done manually, making it difficult to use in industrial solutions.

Stroulia et al. [7, 8] outline an overall process of legacy migration using the CelLEST method. The user's interactions with a legacy system are reverse engineered and the task-specific segments of this interaction are wrapped in new web-accessible front-ends. The process consists of three steps:

1. Collect system-user interaction traces using specially instrumented, non-intrusive middleware.
2. Reverse engineer the dynamic behaviour of the system interface in terms of the screens it presents to the user and user navigation through them.
3. Analyze the task-specific navigation paths to extract a model of the user's task of interest, in terms of the interface navigation and the information exchange it implies.

CelLEST has been demonstrated on infoMcGill, a legacy application of McGill University. The case study was done by a single person familiar with the legacy system. This is a disadvantage of the approach, since a person familiar with the legacy code cannot always be found. CelLEST does not require any understanding of legacy code, modeling instead the tasks accomplished by legacy application users based on traces of their interactions. Although this code-independence is an advantage, it can be complicated to use the approach when the number of interactions between the users and the legacy application is large.

In Sneed and Sneed [9], an interface is constructed wrapping the navigation and execution of a legacy system for a standard web browser. Individual blocks of code are wrapped and reused as web services using a seven step process: Function Mining, Function Wrapping, XML Schema Creation, Server Stub Generation, Client Class Generation, Server Linking, and Web Service Binding. Functions of the legacy code are extracted based on information provided by the SoftWrap tool, however there must be a study done before choosing the functions.

4.2 Comparison of the techniques

Table 1 summarizes the wrapping approaches according to our comparison criteria. All the techniques have advantages and disadvantages. However, the approach presented by Canfora [5] is largely manual, which makes it the least preferred approach. It is difficult to evaluate the complexity of the approaches, since all techniques depend a great deal on the size of the legacy system. In general however the complexity of the wrapping techniques is low, since there is no deep analysis of the legacy system and only the interface is exposed as web services.

Ref.	Legacy System Type	Degree of complexity	Analysis Depth	Process Adaptability	Tool support	Degree of coverage	Maturity Level
Sneed [3,4]	Legacy programs	NA	Depends on business rules in the legacy code	Code stripping	Automated	Complete	Case study
Stroulia et al. [7,8]	Code independent	Depends on the legacy application	User interactions with legacy application (code indep.)	Model interface behaviour of the system & users	Semi-automated	Complete	Case study
Canfora et al. [5,6]	Interactive legacy system	NA	Use cases of legacy app.	NA	Manual	Complete	Case study (Email client)
Sneed & Sneed [9]	Individual blocks of code	NA	NA	NA	Automated	Complete	Case study

Table 1. Summary of Wrapping Techniques

5 Redevelopment Strategies

In this study we use the term redevelopment to refer to reengineering approaches. Reengineering is the analysis and adjustment of an application in order to represent it in a new form. Reengineering can include activities such as reverse engineering, restructuring, redesigning, and re-implementing software. The following approaches use reverse engineering and reengineering to add new SOA functionality to existing legacy systems.

There are three main issues in service-oriented reengineering: service identification, service packaging, and service deployment. Identification of services from a legacy system is not an easy task. Software reengineering can play an important role in migration to the service-oriented environment. It is especially applicable to legacy systems with the following characteristics:

1. The legacy system needs to be migrated to a distributed environment and can be wrapped and exposed as a Web Service.
2. The legacy system has embedded reusable and reliable functionality with valuable business logic.
3. Some of the components in the legacy system are more maintainable than the whole legacy system.
4. The embedded functionality is useful to be exposed as independent services.
5. Target components need to run on different platforms or vendor products.
6. Some of the legacy components can be replaced gradually without affecting the service consumer.

5.1 Overview of redevelopment techniques

Chung et al. (2005) [10] describe a project in which a legacy theorem proof checking and derivation tool called Bertie3 is reengineered to service-oriented architecture, resulting in a new tool called Service-Oriented Bertie (SoBertie) that provides the core capabilities of Bertie3 as web services.

Chung et al. (2007) [11] present a service-oriented software reengineering (SoSR) methodology designed for applying SOA to legacy systems. The SoSR methodology, a synthesis of best practices, is architecture-centric, service-oriented,

role-specific, and model-driven. It is conceptualized from a three service-participants model, a 4+1 view model, and RACI (responsibility assignment) charts. Although there are no full examples using SoSR as yet, a case study of the purchasing department of a retail store inventory system called GMPO is presented.

Distante et al. [12] present a holistic approach to redesigning legacy applications for the Web using the Ubiquitous Web Applications Design Framework (UWA) and an extended version of its Transaction Design Model (UWAT+). It consists of design recovery technologies for the legacy application and forward design methods for Web-based systems. The process used to produce the UWA/UWAT+ conceptual design of the new application consists of three steps: requirements elicitation, reverse engineering, and forward engineering.

Chen et al. [13] use feature analysis to support service-oriented reengineering. Feature analysis includes identifying system features, constructing a feature model to organize the identified features, and identifying their implementation in the legacy system through feature location techniques.

The authors used a library management information system (MIS) in a digital campus as a case study. The MIS is analyzed using a top-down technique of domain decomposition and feature analysis. Several services are identified along with their features. The implementation of the identified services is then generated by a web services wrapper tool. The tool is able to generate the glue code for Web Services and the related source code of Web Service methods.

Cuadrado et al. [14] propose a process for recovering legacy system architecture in order to identify the plan to be carried out in modernizing the legacy system. Theirs is a white-box approach based on modifying the existing legacy code. It uses a three step process consisting of legacy architecture recovery, creating an evolution plan, and executing the plan. Architecture recovery supports the creation of proper documentation. The evolution plan consists of four phases: architecture selection, definition of evolution cycles, planning of the cycles, and a preliminary feasibility check. The process is completed by execution of the plan.

5.2 Comparison of the techniques

Table 2 summarizes the characteristics of the redevelopment approaches using our comparison criteria. For these techniques it was difficult to identify the degree of process adaptability. While each technique uses a different approach to identifying the legacy code with business value, all of them provide tool support. All of the techniques except Chung et al. (2005) [10] provide a case study.

6 Migration Strategies

In migration strategies, legacy code is identified, decoupled, and extracted using approaches similar to those used in wrapping and redevelopment. User interfaces are then reengineered to be compatible with an SOA structure. Migration strategies incorporate both redevelopment and wrapping and aim to produce a system with an improved SOA-compatible design. It is not always obvious how to distinguish migration approaches from wrapping and redevelopment techniques. Here we use the term migration when referring to any approach which moves the entire legacy system and its core framework to the new environment.

Ref.	Legacy System Type	Degree of complexity	Analysis Depth	Process Adaptability	Tool support	Degree of coverage	Maturity Level
Chung et al. (2005) [10]	Logic derivation program	Moderate	Dependent	NA	Yes	Complete	Proposed Idea
Chung et al. (2007) [11]	Interactive legacy system	Moderate	Reverse software engineering & forward soft. eng.	Reverse software engineering	Yes	Complete	Case study
Distante et al [12]	Windows stand-alone application	Time consuming	Design recovery & forward design methods	Web transact. & navigation mode	Yes	Complete	Case study
Chen et al. [13]	Technical environ.	Dependent	Source code identification	NA	Yes	Complete	Case study (Lib. MIS)
Cuadrado et al. [14]	Dependent	Dependent	Detailed description of legacy system	NA	Eclipse TPTP & Omondo UML	Complete	Case study (Medical imaging sys.)

Table 2. Summary of Redevelopment Techniques

6.1 Overview of migration techniques

Aversan et al. [15] present a case study in which a COBOL system is migrated to a web-based service oriented architecture. The legacy system is divided into user interface and server (application logic and database). The user interface is migrated to a Web browser shell using Microsoft Active Server Pages and the VBScript scripting language and the MORPH approach has been used to map the components of the existing interface onto the new Web based interface. The server is wrapped and integrated into the new web-enabled system with dynamic load libraries written in Microfocus Object COBOL, loaded into Microsoft Internet Information Server (IIS), and accessed by the ASP pages.

O'Brien et al. [16] present a strategy that identifies and uses legacy components as services. Architecture reconstruction is used to identify dependencies between components for migration to services and thus provide an organization with a better understanding for their decision-making process.

Lewis et al. [17, 18] and Smith [19] discuss a migration technique called the Service-Oriented Migration and Reuse Technique (SMART) that helps organizations analyze legacy systems to decide whether their functionality can reasonably be exposed as services in a Service-Oriented Architecture (SOA). SMART considers the specific interactions that will be required by the target SOA and any changes that must be made to the legacy components. It gathers a wide range of information about legacy components, the target SOA, and potential services to produce a service migration strategy as its primary artifact.

Z. Zhang et al. [20] propose a reengineering approach that applies a hierarchical clustering algorithm to understand the legacy code in order to extract it for web service construction. The clustering technique is used to extract independent services from legacy code. The technique supports service identification and packaging, providing functional legacy code as web services.

J. Zhang et al. [21] discuss a current project on the design and development of pass-through authentication (PTA) web-services for on-line electronic payment applications. The application is an on-line synchronous/asynchronous payment

Ref.	Legacy System Type	Degree of complexity	Analysis Depth	Process Adaptability	Tool support	Degree of coverage	Maturity Level
Aversano et al. [15]	COBOL Program	NA	Static analysis	Legacy programs essentially unchanged	Yes	Complete	Pilot project
O'Brien et al. [16]	System Independent	NA	Architecture Reconstruction	Information on legacy system is gathered, legacy code is unchanged	Yes	Identification & reuse of legacy compon. as services	Case Study On C++ Code
Lewis et al. [17,19] Smith [18]	Program Independent	Depends on legacy system	Architecture reconstruction & detailed analysis of the target SOA	Legacy system characteristics, architecture, and code is gathered	Yes	Complete	Set of Guidelines
Zhang et al. [20]	Object-oriented program	NA	Hierarchical clustering to identify services	Domain analysis is used to identify the business logic	Yes	Complete	Case Study
Zhang et al. [21]	Legacy e-Payment systems	NA	Original legacy system integrated into target system	NA	NA	Complete	Case Study
Cetin et al. [22]	Program Independent	NA	Legacy system is analyzed	If change is needed, legacy components modified or replaced	Yes	Complete	Case Study
Marchetto & Ricca [23]	Java Application	Moderate	UML Use Case diagram	The internal structure can be changed if needed	Yes	Complete	Case Study

Table 3. Summary of Migration Techniques

processing application that can perform real-time or batched payment transactions. Although this approach exposes business logic in legacy code as services, the main concern is not to achieve SOA architecture, rather to expose the legacy system's functionality as web services.

Cetin et al. [22] propose a mashup migration strategy, addressing both the behavioural and the architectural aspects of the migration process. Their method consists of six steps: model the target enterprise business requirements; analyze the existing legacy system; map the target enterprise model to legacy components and identify services; design a concrete mashup server architecture; define service level agreements; and implement and deploy services. The main idea is the integration of the the legacy system at the presentation layer, which requires re-inventing the popular mashup technology of Web 2.0 at the enterprise level.

Marchetto and Ricca [23] present a stepwise approach for Java where one candidate service is migrated to a Web service in each migration step. The goal of their approach is to obtain a "preliminary" service-oriented implementation of the original system, not necessarily the "best one".

6.2 Comparison of the techniques

Table 3 summarizes the migration approaches according to our comparison criteria. Each of the migration techniques uses a different method to achieve the result. All the techniques provide a case study to support their claims, except for SMART [17, 19, 18], which is introduced as a set of guidelines to direct the process of migration. Since SMART is legacy system independent, it could be used in any of the other techniques to evaluate the legacy system involved in

Strategy	Advantages	Disadvantages
Replacement	Reduce maintenance Improve business functions	Time consuming Expensive Experienced resources needed
Wrapping	Fast	Inflexible Difficult maintenance
Redevelopment	Increase agility Flexibility Reduced cost	Source code needed Original requirements needed
Migration	Stable environment Tools availability	Time consuming Experienced resources needed Source code needed

Table 4. Summary of modernization strategies

the migration. While the benefits of the migration strategy seem to be well understood, there is still no general migration technique that can be applied that solves all of the problems that a developer may face.

7 Choosing A Strategy

When choosing a strategy, a variety of aspects come into play. Table 4 summarizes an initial set of strengths and weaknesses for each strategy. Other possible aspects might include the type of company that owns the software and the kind of stakeholders that use the systems.

Two or more modernization strategies can be combined to achieve the required goal depending on the advantages and disadvantages of each strategy in the context. It is not always easy to reuse legacy system components and expose them as services. In some situations, exposing them as services will have a higher risk and a higher cost than replacing them entirely with a new SOA architecture. There is no perfect solution to the problem of modernizing a legacy system. The choice of strategy depends entirely on the goals for the SOA architecture, the available budget and resources and the time needed to complete the project.

8 Conclusions

While modernizing legacy systems for service-oriented architecture has clear potential benefits, it is important to choose the appropriate modernization strategy. In this study, we have reviewed the four main approaches in the literature for migrating legacy systems to SOA: replacement, wrapping, redevelopment and migration, and highlighted the major techniques of each kind described in the literature. While no one approach applies to every situation, by comparing the maturity, applicability, strengths and weaknesses of each of them, we can better understand how to choose among strategies for any given project.

Several future directions are worth investigating. The main challenge in modernizing a legacy system is finding the business services in legacy code, and there is a need for better tools and techniques to identify business value in large code bases. There is a need for objective metrics to evaluate the techniques, and most approaches do not discuss measuring the quality attributes of the resulting services, such as security, performance and reliability. Finally, while web services is one technology to implement SOA, it would interesting to examine others.

References

1. J.Chatarji: Introduction to service-oriented architecture (SOA) (2004)
2. Comella-Dorda, S., Wallnau, K.C., Seacord, R.C., Robert, J.E.: A survey of black-box modernization approaches for information systems. In: ICSM. (2000) 173–183
3. Sneed, H.M.: Integrating legacy software into a service oriented architecture. In: CSMR. (2006) 3–14
4. Sneed, H.M.: Wrapping legacy software for reuse in a SOA. Technical report (2005)
5. Canfora, G., Fasolino, A.R., Frattolillo, G., Tramontana, P.: Migrating interactive legacy systems to web services. In: CSMR. (2006) 24–36
6. Canfora, G., Fasolino, A.R., Frattolillo, G., Tramontana, P.: A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. J. Systems and Software **81**(4) (2008) 463–480
7. Stroulia, E., El-Ramly, M., Sorenson, P.G., Penner, R.: Legacy systems migration in CelLEST. In: ICSE Posters. (2000) 790
8. Stroulia, E., El-Ramly, M., Sorenson, P.G.: From legacy to web through interaction modeling. In: ICSM. (2002) 320–329
9. Sneed, H.M., Sneed, S.H.: Creating web services from legacy host programs. In: WSE. (2003) 59–65
10. Chung, S., Young, P.S., Nelson, J.: Service-oriented software reengineering: Bertie3 as web services. In: ICWS. (2005) 837–838
11. Chung, S., An, J.B.C., Davalos, S.: Service-oriented software reengineering: SoSR. In: HICSS-40. (2007) 172c
12. Distante, D., Tilley, S.R., Canfora, G.: Towards a holistic approach to redesigning legacy applications for the web with uwat. In: CSMR. (2006) 295–299
13. Chen, F., Li, S., Chu, W.C.C.: Feature analysis for service-oriented reengineering. In: APSEC. (2005) 201–208
14. Cuadrado, F., García, B., Dueñas, J.C., G., H.A.P.: A case study on software evolution towards service-oriented architecture. In: AINA. (2008) 1399–1404
15. Aversano, L., Canfora, G., Cimitile, A., Lucia, A.D.: Migrating legacy systems to the web: An experience report. In: CSMR. (2001) 148–157
16. O'Brien, L., Smith, D.B., Lewis, G.A.: Supporting migration to services using software architecture reconstruction. In: STEP. (2005) 81–91
17. Lewis, G.A., Morris, E.J., Smith, D.B.: Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In: CSMR. (2006) 15–23
18. Lewis, G.A., Morris, E.J., Smith, D.B., O'Brien, L.: Service-oriented migration and reuse technique (smart). In: STEP. (2005) 222–229
19. Smith, D.B.: Migration of legacy assets to service-oriented architecture environments. In: ICSE Companion. (2007) 174–175
20. Zhang, Z., Yang, H.: Incubating services in legacy systems for architectural migration. In: APSEC. (2004) 196–203
21. Zhang, J., Chung, J.Y., Chang, C.K.: Migration to web services oriented architecture: a case study. In: SAC. (2004) 1624–1628
22. Cetin, S., Altintas, N.I., Oguztuzun, H., Dogru, A.H., Tufekci, O., Suloglu, S.: Legacy migration to service-oriented computing with mashups. In: ICSEA. (2007) 21c
23. Marchetto, A., Ricca, F.: From objects to services: toward a stepwise migration approach for java applications. STTT **11**(6) (2009) 427–440