Chapter 4
Reengineering and Wrapping Legacy Modules for Reuse as Web Services: Motivation, Method, Tools, and Case Studies

Harry M. Sneed
ANECON GmbH, Austria

ABSTRACT

In almost every IT-user shop where a mainframe computer is still in operation, there are hundreds if not thousands of legacy code modules with several million lines of code. Most of them are written in COBOL, but also many in PL/I and 4GL languages like Natural and PowerBuilder. Some are running online in dialog mode while others run in batch mode as background jobs. Many of these users would now like to migrate to a service-oriented architecture using the mainframe as a hub where a wide assortment of Web services are available to the client business units to be built into their new business processes. However, they are inhibited in doing so by the high costs of such a migration. The biggest costs lie in developing the Web services from scratch or adapting those that can be found on the market.

These high costs could be avoided if the user were able to reuse the software he already has. Most of the elementary business functions required by the new business processes already exist inside the old legacy modules. If they could be extracted and reused, the user would save the costs of having to develop them. However, their current architecture prohibits their reuse. They have no clearly defined interface, they are dependent on an old teleprocessing monitor or batch framework; they are intertwined with the other modules around them and their data is more often incompatible with the data types used for interacting with Web services. That means the modules have to be reengineered before being reused.

This chapter deals with how this reengineering can be done. The goal is to make modular, flexible, and independent Web services from the monolithic, rigid, and dependent legacy modules. The methods used to achieve this goal are static analysis, code restructuring, code stripping, code transformation, and code...
Reengineering and Wrapping Legacy Modules for Reuse as Web Services

INTRODUCTION

There are many reasons for users to want to migrate to a service-oriented architecture. There are both business and technical reasons. The business reasons are among others:

- To become faster and more flexible in responding to user requirements,
- To give the users a wide range selection of ready-made services,
- To unify common business functionality,
- To cut the costs of maintaining an oversized software base.

In short, the business goal is to provide more business functionality in shorter time with fewer costs.

The technical reasons for migrating are among other:

- To reduce the complexity of the existing IT-landscape,
- To eliminate the many redundancies associated with a silo-type software architecture,
- To reduce the amount of software which has to be maintained,
- To unify the diverse software solutions in a common architectural mode.

In short, the technical goal is to simplify and unify the user’s software base, thus reducing the costs of maintaining it.

The Case for Reusing Legacy Modules as Web Services

In moving from traditional silo type architectures of loosely coupled application systems to a service-oriented architecture, users have a number of options (Sneed, 2007). One is to purchase a standard set of Web services from a software vendor and to adapt their business processes to them. This requires them to give up much of their local rules and specific business logic. They can customize the overall processes, but they have to accept the detailed data and algorithms provided by the vendor. A variation of this option is to use the standard Web services on demand, thus avoiding the costs of owing them. Cloud services promise to be a cheap and readily accessible. They can be incorporated into the user business processes and integrated with the existing components. From an economic point of view this is a tempting solution, but it does not alter the fact that the user has to adapt his way of working to fit these off-the-shelf solutions (Yau & Ho, 2011).

A second option is to take over Web services from the open source community and to modify them to fit his needs. This give him the possibility of inserting his own rules and detailed logic, but it requires him to have specialists who can work with the open source. If such specialists are not available, this approach can become very expensive depending on the degree of change. Besides, it is error prone. Any alteration of the open source can lead to a chain of errors, which if the developers are not familiar with the source, will be difficult to correct. It appears to be cheap,