Chapter 6
The SOA Frontier: Experiences with Three Migration Approaches

Juan M. Rodríguez
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Marco Crasso
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Cristian Mateos
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Alejandro Zunino
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Marcelo Campo
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Gonzalo Salvatierra
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

ABSTRACT

Service Oriented Architecture (SOA) and Web Services are the current trend to integrate large and distributed systems, which is a common situation in both the business and government worlds. However, within these worlds, systems are commonly written in COBOL because they were developed several decades ago. Therefore, migration of COBOL systems into service-oriented architectures becomes a necessity. Two main approaches are used to migrate COBOL systems to SOA systems: direct and indirect migration. Direct migration implies wrapping the current COBOL routines of a system with a software layer developed under a newer platform that can be used to offer Web Services. In contrast, indirect migration requires re-designing and re-implementing the COBOL routines’ functionality using a newer platform as well. In this chapter, the authors propose a novel migration approach, which takes the best of the two previous approaches. To assess the advantages and disadvantages of these approaches, this chapter presents a case study from a government agency COBOL system that has been migrated to a Web services-based system using the three approaches. As a result of having these migration attempts, the authors present the trade-off between direct and indirect migration, the resulting service interfaces quality, and the migration costs. These results also show that this new migration approach offers a good balance to the above trade-off, which makes the approach applicable to similar COBOL migration scenarios.

DOI: 10.4018/978-1-4666-2488-7.ch006
INTRODUCTION

Information systems were adopted by enterprises several decades ago, and they have been used ever since. As a result, operationally, most enterprises rely on old, out-of-date systems. These systems are known as legacy systems. Well-known examples of this kind of systems are COBOL systems because, according to Gartner consulting\(^1\), there are over 200 billion lines of operative COBOL still running worldwide. Furthermore, since enterprises’ goals and context vary, in time these systems have suffered modifications to be kept suitable for the enterprises. For example, nowadays it is impossible to conceive a bank that does not offer Home Banking services, yet most bank systems were originally written in COBOL. Therefore, it is common to find a 50 year old technology, such as COBOL, working alongside with the most modern technologies (Lewis, et al., 2011), like .Net, AJAX, or JEE, in the same enterprise. As a result of this situation, enterprises have to face high costs for maintaining their systems. This is mainly because these systems usually run on mainframes that must be rented. In addition, there is the necessity of hiring developers specialized in old technologies for updating the system functionality, which is both expensive and rather difficult.

Taking these facts into consideration, many enterprises opt for modernizing their legacy systems using a newer technology. This process is called migration. Currently, a common target paradigm for migrating legacy systems is SOA (Service-Oriented Architecture) (Bichler & Lin, 2006; Ionita, et al., 2008). In SOA, systems are built using independent functionalities called services that can be invoked remotely. Services are offered as platform-agnostic functionalities that can be used by any system inside or outside their owner organization. To ensure platform independence, Web Services technologies are the commonest way of implementing SOA systems since the former relies on well-known Internet protocols, such as SOAP and HTTP (Erickson & Siau, 2008). Therefore, migrating legacy systems to Web Services technologies is a fairly common practice. However, there is no standard recipe to effectively migrate legacy systems to SOA.

Recent literature (Li, et al., 2007) proposes that a legacy system can be migrated by following two approaches. The first approach, called direct migration, consists in wrapping a legacy system with a software layer that exposes the original system programs as Web Services. This approach is known to be cheap and fast, but it has the disadvantage that the legacy system is not replaced by a new one. Instead, the enterprise obtains two systems to maintain: a legacy system, and its newly-built SOA layer. On the other hand, the approach called indirect migration is based on re-implementing the legacy system using a modern platform. This approach is expensive and time consuming because not only the system should be re-implemented and re-tested, but also in some cases, the business logic embodied in the legacy system should be reverse-engineered. This happens because system documentation could have been lost or not kept up-to-date. The result of an indirect migration is not only an improved version of the system, but also updated documentation for future reference.

From the SOA point of view, an important difference between direct migration and indirect migration is the quality of the SOA frontier that is the set of Web Services exposed to potential consumers as a result of migrating a legacy system. The SOA frontier quality is a very important factor for the success of a SOA system (Blake & Nowlan, 2008; Beaton, et al., 2008; Rodriguez, et al., 2010b) because, as Figure 1 depicts, SOA frontier is used for both service registries and consumer. Service registries use the SOA frontier for indexing services and, then, allowing service consumers to search for them. In contrast, service consumers need the SOA frontier to understand and invoke the services. In addition, a SOA system success can be measure by how many service consumers it