Chapter 1.26
Software Modernization of Legacy Systems for Web Services Interoperability

Chia-Chu Chiang
University of Arkansas at Little Rock, USA

INTRODUCTION

Software maintenance is an inevitable process due to program evolution (Lehman & Belady, 1985). Adaptive maintenance (Schenidewind, 1987) is an activity used to adapt software to new environments or new requirements due to the evolving needs of new platforms, new operating systems, new software, and evolving business requirements. For example, companies have been adapting their legacy systems to Web-enabling environments of doing business that could not have been imagined even a decade ago (Khosrow-Pour & Herman, 2001; Werthner & Ricci, 2004).

To understand software modernization of legacy systems for Web services, it is necessary to address how legacy integration has evolved from centralized computing to distributed, component-based computing due to the advent and widespread use of object-oriented and client-server technologies. Legacy systems were typically developed on a centralized, terminal-to-host architecture. Users usually accessed their legacy systems through terminals that included character-based menus and data entry screens. Consequently, legacy systems built on the central mainframe are inaccessible remotely without adaptations.

Component-based middleware technologies, such as Java RMI, common object request broker architecture (CORBA), and component object model/distributed component object model (COM/DCOM), provide solutions to support the interoperability of legacy systems in a heterogeneous and distributed environment (Chiang, 2001). Unfortunately, the technologies have proved to be insufficient in application integration solutions for several reasons (Stal, 2002). Although the technologies share common communication architectural foundations, the implementation of each technology differs in several aspects, including the object models provided, the communication protocols, and data marshaling/demarshaling. Due to the proprietary implementations of the technologies, they do not interoperate well with each other. Obviously, existing component-based middleware only partially solves the interoperability problems of legacy systems. More effort is still required to make the legacy systems totally interoperable in a heterogeneous and distributed environment.

BACKGROUND

Web services have been widely considered as a better solution to legacy integration for software interoperability using open standards that include extensible markup language (XML), the simple object access protocol (SOAP), the Web services description language (WSDL), and the universal description, discovery, and integration (UDDI) (Chung, Lin, & Mathieu, 2003; Stal, 2002; Zhang & Yang, 2004). Service requesters and providers follow the Web service standards for message exchanges. When a service provider has a service for public exposure, it must write a description of the service in WSDL and register the service description with UDDI to a global repository. A service requester can then query the repository using UDDI to retrieve the service description. The service requester uses the service description in WSDL to send requests, and the service provider replies to the requests under SOAP.

LEGACY MODERNIZATION FOR WEB SERVICES AND CHALLENGES

There are three main reasons for modernizing legacy systems: to reduce the system evolution risk, to recoup the investment on the systems, and to make the system distributed and scalable for business-to-consumer and business-to-business, as well as making it highly available to Web users.

Companies usually have two approaches to turn their legacy systems into Web services: wrapping and reengineering. Wrapping provides a cost-effective way to integrate legacy systems with Web services into a heterogeneously distributed computing environment. Unfortunately, the wrapping approach requires the whole legacy system to be exposed to the public as a Web service, which fails to properly abstract the system (Vinoski, 2002; Vogels, 2003). Furthermore, the wrapping approach increases the difficulty of maintaining the legacy system in the long run. Thus, the wrapping approach is generally a temporary solution, rather than a strategic one. The reengineering approach applies reverse engineering techniques to legacy systems to recover business rules, and develop Web services from the extracted business rules. This approach streamlines legacy systems but is highly dependent on the success of recovery on the business rules from legacy systems.

Wrapping legacy systems for Web services can be performed through wrappers or adapters. A wrapper is built to encapsulate a legacy system and provide access to the legacy system through the encapsulation layer. This layer exposes only the methods with parameter attributes to remote service requesters. In addition, the wrapper must resolve the incompatible communication issues between the legacy systems and the Web server using SOAP/XML messaging. Therefore, programmers are required to write a wrapper to reconcile the issues, as well as a WSDL for public exposure. Unfortunately, a wrapper is difficult to maintain, inefficient, and error-prone (Engelen, Gupta, & Pant, 2003). A sample Web service architecture via a wrapper is shown in Figure 1.

Turning legacy systems in middleware-based components into Web services is slightly different from the technique described above. Because the legacy system has already been wrapped in middleware, companies may be unwilling to unwrap their systems in order to turn the system into a Web service. Fortunately, there are Web services toolkits available to turn middleware-based componentized legacy systems into Web services (Engelen, Gupta, & Pant, 2003). First, the toolkits translate the interface definition of a