Towards a Web-centric Legacy System Migration Framework

Ying Zou and Kostas Kontogiannis

Dept. of Electrical & Computer Engineering
University of Waterloo
Waterloo, ON, N2L 3G1, Canada
{yzou, kostas}@swen.uwaterloo.ca

ABSTRACT

With the rapid growth of B2B and e-Commerce activities, the "Service Web" has emerged as a promising framework to address the issues related to process integration. In this paper, we present a stack of interrelated protocols, such as SOAP, WSDL, and UDDI, to support service integration using the Web as the medium. Specifically, we aim for a framework that allows for the identification of reusable business logic in large legacy systems in the form of major legacy components, the identification of interfaces between the legacy components and the rest of the legacy system, the automatic generation of CORBA wrappers to enable for remote access, and finally, the seamless interoperation with Web services via HTTP based on the SOAP messaging mechanism.

1 INTRODUCTION

With the rapid global adoption of B2B and e-Commerce activities worldwide, the concept of the Service Web [8] aims at empowering the business process integration over the Web. To facilitate the use of the Web for business process integration and collaboration between trading partners, the "Service Web" builds on the top of components referred to as Web services [4]. Such Web services offer specific business related functionality, reside in the application servers, can be programmatically integrated with other systems, and perform interactive tasks involving multiple steps on a user's behalf.

To allow for Web services to interoperate, a common industry standard, such as SOAP (Simple Object Access Protocol) [1, 5], WSDL (Web Service Description Language) [2], and UDDI (Universal Description, Discovery, Integration) [3] has been proposed. In a nutshell, SOAP, encoded in XML and HTTP, is a message passing mechanism and serves as a uniform invocation mechanism between Web Services. The WSDL describes the interface points of the Web services that are further indexed in searchable UDDI [6].

For many organizations, the task of renovating their legacy systems towards such a new architecture advocated by the Service Web is not an easy one, due to the significant risk and investment required to re-implement large portions of their legacy software systems to comply with network-centric technologies. An alternative technique to renovation is the reuse of the business processes in the form of wrapped components.

In this paper, we present a framework to migrate legacy systems in a new Web enabled environment where they are considered as distributed services. It consists of two major steps. In the first step, the monolithic legacy systems are decomposed into software components, and migrated from their standalone form into a CORBA distributed objects by automatically generating wrappers. In the second step, a layer between CORBA objects and the SOAP protocol wraps the CORBA object requests with SOAP messaging. In such a way, SOAP becomes a unified mechanism to tie the migrant legacy services with other systems and to coordinate the invocation between the different service entities.

This paper is organized as follows. Section 2 provides a brief overview of standards for the support of Service Web. Section 3 discusses the concrete approach for the migration of legacy system into Service Web. In the end, section 4 gives conclusion.

2 STANDARDS FOR SERVICE WEB

A large-scale legacy system can be divided into four layers, including standards and guidelines at lowest level, basic common services, value-added functional services, and mission-specific services [7]. The levels of basic common services and value-added functional services can be reused as common facilitators in an application domain. In this case, it is valuable to migrate such blocks into the Web enabled environments as Web Services to maximize the accessibility and reusability of software components.

To define the infrastructure of the Service Web, a stack of interrelated standards, such as SOAP, UDDI and, WSDL are presented. Such standards are denoted in XML, and therefore are extensible, flexible and, can be rich in semantic content. Moreover, they can be transported via the HTTP over the existing Web

infrastructure. An overview of these technologies is provided below.

SOAP

SOAP provides a simple and lightweight mechanism to exchange structured and typed information, such as commands and parameters, between HTTP clients and servers. The SOAP protocol consists of three parts [5] namely, the SOAP envelope, the SOAP encoding rules, and the SOAP RPC representation.

The SOAP envelope carried by HTTP conveys the message content for the application-to-application communication, for example, the request or response in a client/server model.

The SOAP encoding style specifies the rule for serialization of application-defined data types. The encoding schema specifies simple (scalar) types and compound types. It is a generalization of the common features in type systems, including programming languages, databases and semi-structure data.

The SOAP RPC representation defines a binding between SOAP and RPC to make remote procedure calls and responses.

SOAP is an application level protocol standard that utilizes HTTP, and therefore enables for pervasive access. Moreover, SOAP defines a simple, yet flexible messaging mechanism without the complex object model as CORBA and DCOM. In a nutshell, SOAP is responsible for the delivery of Web service requests/responses between clients and servers. Once the message conformance to SOAP is received by the server, the concrete implementation, invocation and execution of Web Services are based on the distributed object technologies, such as CORBA, Java RMI, Java Servlet, Java JSP, or RPC. The SOAP run-time engine runs in an application server, marshalling or unmarshalling the SOAP based requests and responses.

WSDL

WSDL is an evolving specification partially based on Microsoft's SOAP (Simple Object Access Protocol) Contract Language and IBM's NASSL (Network Accessible Service Specification Language) [4]. WSDL precisely describes programmatic interfaces of Web services in XML. Meanwhile, WSDL can be used to generate the proxy objects for the Web services in the client and server sides [6]. The language is limited to message formats or network protocols that conform to SOAP 1.1, HTTP get/post and MIME (Multipurpose Internet Mail Extensions).

UDDI

The UDDI is a Business Registry standard for indexing Web Services, and enables businesses to find preferred services and transact with one another. In this context, Web Services can be registered in a standard way in service repositories and be located at run time by client processes. The information required by UDDI to register a service is structured in three sections [3] namely, white pages containing company descriptions, contact information, and business ID numbers; yellow pages with companies organized into business categories, such as products, geographic regions, and industry sectors; and finally green pages, which will provide transaction requirements. The standard is based on extensible markup language (XML), and HTTP and Domain Name System (DNS) protocols. UDDI addresses cross-platform programming issues through its adoption of the Simple Object Access Protocol (SOAP) messaging specifications.

3 MIGRATION OF LEGACY SYSTEM INTO SERVICE WEB

Within the context of re-engineering legacy systems into a Service Web environment, it is important to analyze the legacy software to recover its major functional components that relate to an application domain and specific business logic. Reverse engineering techniques that allow for the identification of such components have been investigated in depth by the research community. Once specific legacy components have been identified through the use of program analysis, their behaviors can be specified in terms of well-defined object oriented interfaces. In the context of our research, a wrapping technique was adopted for the deployment of the identified components into a distributed environment based on CORBA infrastructure. Although CORBA IDL is a platform-independent standard, it requires the communication entities have the ORB brokers installed before the services can inter-operate. This requirement is difficult to meet in the Web environments, due to heterogeneous web-enabled devices, clients, and protocols that are supported in different sites. The SOAP messaging mechanism is the preferred choice for the migrant CORBA services to be accessed in such a wide diversity environment. In the following subsections, our approach for the utilization of these technologies is discussed in more detail.

3.1 IDENTIFING LEGACY COMPONENTS

To identify software components that maximize reusability and flexibility, our approach aims to transform selected procedural code from a legacy component to a new object oriented architecture where the interfaces of the component are clearly defined and the related operations that deliver specific functionality are fully encapsulated in classes and wrappers.

The first step towards the migration of a procedural system to an object-oriented architecture is the selection of possible object classes by analyzing global variable data types as well as, complex data types in formal parameter lists. Analysis of global variables and their corresponding data types is focusing on the identification

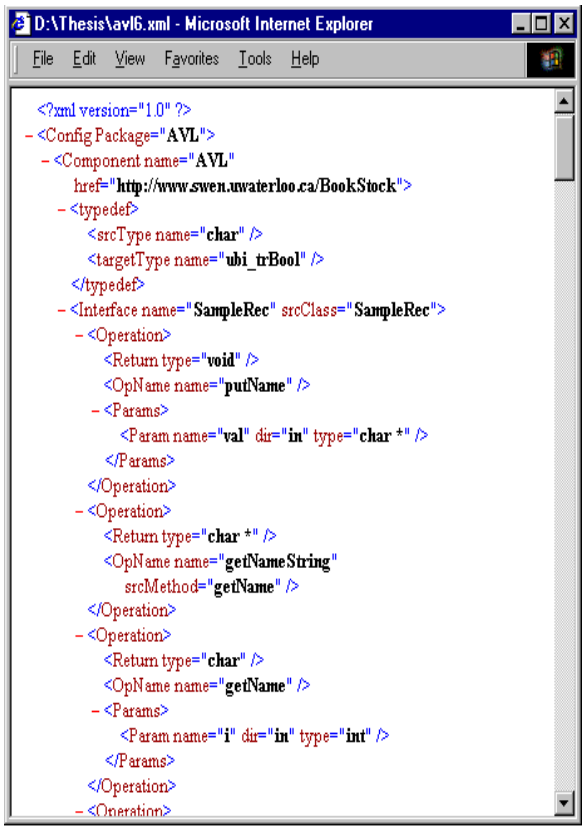

Figure 1: XML Representation of Component Interface

of variables that are globally visible within a module. A module is defined as a consecutive set of program statements that deliver particular functionality and can be referred to by an aggregate name [9], for example a file or a collection of files. Any variable that is shared or is visible (that is, it can be fetched and stored) by all the components of a module is considered as a global variable. Clustering techniques and architectural recovery techniques presented in [10, 11] can be used in order to obtain an initial decomposition of a large system in terms of module components. For each variable its corresponding type can be extracted from the Abstract Syntax Tree, and a candidate object class can be generated.

Once an initial set of candidate classes is identified, then the methods to be attached to the classes can be identified by a set of evidence models, such as function return type analysis, state modification analysis, use analysis and metrics analysis.

This task can be automated to a large extend using a number of different software analysis techniques. However, no matter how sophisticated the analysis techniques are, user assistance and guidance is crucial on obtaining a viable and efficient object model. Significant domain information can be utilized by the user to guide the discovery process, locate the common functionality in an application domain and obtain a better and more suitable object model.

3.2 MIGRATING COMPONENTS INTO NETWORK CENTRIC ENVIRONMENTS

Once a software component has been extracted from a legacy system, its interface can be extracted and represented in XML. The CORBA IDL and object wrappers can be automatically generated by the wrapper generator from the XML-based specification.

Representation of Legacy Services

To wrap a legacy service, the first step is to determine its interface. The interface description allows the implementation details inside the component to be hidden from its external clients. In general, the interface of a software component may contain the information related to data types, references to external specifications that point to related components, descriptions of public attributes and methods, and return types and parameters that specify input and output.

The representation of a component interface can be independent of a specific programming language. OMG IDL provides such a language-independent interface description for distributed CORBA components. However, in order to automatically generate wrappers, a specialized IDL parser is required to access the interface information.

In our approach, the XML interface representation of a software component is adopted in order to encode interface information. Figure 1 illustrates an XML based component interface representation. It consists of several aggregated elements, such as data type definitions, interface operation definitions, and interface data member definitions. The data type definition section publishes the data types in the component other than those in its defined interface. The interface data member definition declares the accessor and mutator methods associated with a data member. Such specification aims on automatic generation of OMG IDLs and CORBA wrappers. Furthermore, with the XML specification, additional information such as, reflection information, URL address, pre- and post-conditions, and performance characteristics can be easily denoted for a given component.

Automatic Generation of OMG IDL and CORBA Wrapper

Once the interface is defined, the process of generating CORBA wrappers is similar for every identified legacy component that is being migrated. The wrappers implement message passing between the calling and the called objects, and redirect method invocations to the actual component services. The concrete process to accomplish wrapping is accomplished in terms of three major steps.

The first step focuses on automatic generation of

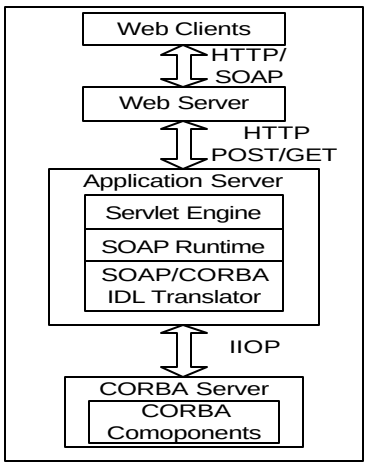

Figure 2: SOAP Wrapping Structure

CORBA IDL interface for the identified components. In the XML interface specification, the signatures are indicated in terms of C++ types. Specifically, the IDL generator reads the XML script and converts the interface descriptions to IDL types, conforming to the mapping rules of the OMG IDL specification, and writes IDL style interface in a new file.

The complete mapping of basic data types is given in [13]. For example, C++ "long" is mapped into IDL "long", C++ "char *" to IDL "string". Due to the platform independence of OMG IDL, some of basic C++ data types are not specified, such as "int", which is 16 bits in MS-DOS and Win3.1, but is 32 bits in Win32. For the identified component, we assume it works under a 32 bit operating system.

The second step deals with the CORBA IDL compiler to translate the given IDL specification into a language specific (e.g. C++), client-side stub classes and server-side skeleton classes. Client stub classes and server skeleton classes are generated automatically from the corresponding IDL specification. The client stub classes are proxies that allow a request invocation to be made via a normal local method call. Server-side skeleton classes allow a request invocation received by the server to be dispatched to the appropriate server-side object. The operations registered in the interface become pure virtual functions in the skeleton class.

The third step focuses on automatic generation of wrapper objects as CORBA objects, by inheriting from the skeleton classes. The wrapper classes encapsulate the standalone C++ object by reference, and incarnate the virtual functions by redirecting them to the encapsulated C++ class methods. The wrapper generator maps the CORBA C++ classes used in CORBA objects into native C++ classes recognized by the identified components using the XML interface specification for the component being wrapped. Since the primary data types are passed by value, the parameters from the client request methods can be directly passed into native C++ operations, according to the same mapping rules from C++ to IDL. For the complex data types, such as class, string, array, the parameters are passed by reference. The signatures of such types in wrapper classes are defined by pointers. In this case, a method is added into to each of CORBA wrapper objects to accomplish the translation from a CORBA C++ wrapper pointer to a native C++ class pointer.

3.3 WRAPPING CORBA OBJECTS WITH SOAP

To enable the universal access to CORBA objects without installing ORB run-time capability on the client-side, especially the thin clients, a lightweight, Web-native approach is critical for the success. In this context, the SOAP protocol is adapted as a uniform messenger between CORBA objects and the rest of Web services.

The layered structure is illustrated in Figure 2. The Web client sends their request, conformance to SOAP, through HTTP to a Web server. The Web server redirects the request to the servlet engine, which is responsible to process the HTTP/POST and HTTP/GET requests. The SOAP run time engine is apt to handle the interpretation and dispatching of SOAP message. At the heart of the structure, a SOAP/CORBA IDL translator, is implemented as a collection of Java Beans to process inbound SOAP requests, which represent invocations to back-end CORBA objects. Internally it encapsulates all the services like SOAP serialization and deserialization, bridges the gap between the SOAP RPC and CORBA IDL and finally, generates the stub to dispatch the requests to the back-end services on the client's behalf. Meanwhile, the WSDL of back-end CORBA Server can be automatically generated from the component interface specification, shown in Figure 1, and can be registered in the UDDI central repository.

4 CONCLUSION

With the occurrence of Service Web, a stack of interrelated standards based on XML has emerged to provide the language-independent data representation and platform-neutral support for the Web services integration.

This paper presents a framework to address the issues on migrating legacy systems into a Web enabled environment. The approach consists of three major steps. First, a legacy system component is leveraged by identifying parts of a legacy system that relates to a specific application functionality. Second, a CORBA wrapper generator facilitates the rapid movement into the distributed environments, and enables the migrant services remote accessible. Finally, a SOAP/CORBA IDL translator layer bridges the gaps among heterogeneous implementation technologies for Web services, and solves the universal integration issues by SOAP and HTTP. A prototype system that supports the framework presented in this paper is currently developed

by the authors at the IBM Toronto Lab, Center for Advanced Studies.

REFERENCES

[1] "Web Services and the Simple Object Access Protocol", http://msdn.microsoft.com/xml/general/soap_webserv.asp.

[2] "Web Services Description Language (WSDL) 1.0", http://www-4.ibm.com/software/devleoper/library/w-wsdl.html.

[3] "Universal Description, Discovery and Integration: UDDI Technical White Paper", http://www.uddi.org.

[4] Arthur Ryman, "Understanding Web Services", IBM Toronto Lab.

[5] "SOAP: Simple Object Access Protocol", http://www-4.ibm.com/software/developer/library/soapv11.html

[6] IBM Web Services Toolkit, http://www.alphaworks.ibm.com/tech/webservicestoolkit.

[7] William J. Brown, et al, "Anti-Patterns: Refactoring Software, Architectures, and Project in Crisis", John Wiley & Sons, Inc, 1998.

[8] Ying Zou, Kostas Kontogiannis, "Localizing and Using Services in Web-Enabled Environments", 2nd International Workshop on Web Site Evolution 2000, Zurich, Switzerland, March 2000.

[9] Shari, Lawrence, Pleeger, "Software Engineering, Theory and Practice", Prentice Hall, 1998.

[10] P. Finnigan, et.al "The Software Bookshelf", IBM Systems Journal, vol.36, No.4, 1997.

[11] R. Selby, V. Basili, "Analyzing Error-Prone System Structure", IEEE Transactions on Software Engineering, Vol.17, No.2, February 1991, pp. 141-152.

[12] Aleksander Slominski, et al, "Design of an XML based Interoperable RMI System: SoapRMI C++/Java 1.1", http://www.extreme.indiana.edu/soap/design/.

[13] Michi Henning, Steve Vinoski, "Advanced CORBA Programming with C++", Addison-Wesley, 1999.