Service-Oriented Architecture: Concepts and Implementation

Mayur R. Mehta
mm07@txstate.edu

Sam Lee
sl20@txstate.edu

Jaymeen R. Shah
js62@txstate.edu
Department of CIS & QMST
Texas State University–San Marcos
San Marcos, Texas 78666, USA

Abstract

In today's increasingly competitive and technology-driven business environment, ability to quickly adapt its business processes in response to both internal and external changes is a must for any organization. Service-Oriented Architecture (SOA), along with its programming models, is one strategic option to accomplish this. SOA enables an organization's business to drive its information systems design by enabling the organization to align its business processes with information technology (IT). SOA is an architectural style that supports integration of business processes as linked services that may be accessed when needed over a network. From the perspective of application developers, SOA is a set of programming models and tools for building, accessing, and assembling services that implement business design. This paper examines the role and benefits of an application development framework built on SOA foundation. In addition, it presents a programming model to build Web services, which is the most common approach used to implement SOA.

Keywords: Service-oriented architecture, Web services, J2EE

1. Introduction

Applications developed in the past were usually standalone applications that performed a specific task(s) such as accounts receivables/payables. Such applications were not integrated with other applications. Use of such standalone applications created islands of automation within an organization, which usually led to duplication of business logic and data redundancy. In most organizations today, both intra- and inter-organizational business processes are enabled by information technology (IT). As business processes usually require the use of functionalities that are embedded within standalone applications that may have been developed during different time periods using different technologies, it has become necessary to integrate these existing standalone applications. Integration of these applications must be such that, if necessary, it should allow the organization to quickly adapt business processes in response to regulatory and/or environmental changes, and also create new business processes by assembling existing business logic that may exist in different applications. The need of these capabilities precludes the use of point-to-point tight integration between applications that are usually difficult and time consuming to manage and adapt. Further, the complexity and cost of managing point-to-point tight integrations tend to increase significantly as the number of applications that are integrated increase. The use of service-oriented architecture (SOA) will enable an organization to use standards-based, vendor agnostic approach to easily and cost-effectively integrate existing disparate standalone applications (Sanchez, 2006). It will make it easier to integrate existing applications with new business functionalities that are implemented as services, and also enhance the ability to integrate internal applications with business partners' applications. Successful integration of internal business applications and their integration with business partners' application will enable organizations to use collaborative business models. Technology integration and use of collaborative business models are considered to be important strategic business technology investments in today's global and competitive business environment (Andriole, 2006).

In the following Section we present an overview of application development models. Then in Section 3, we discuss SOA and the importance of SOA in the current business environment. In Section 4, an overview of Web services is presented along with discussion regarding its role in SOA. An example of successful SOA implementation is briefly discussed in Section 5. In Section 6 we discuss the use of IBM WebSphere Studio Application Developer (the new version of this software is offered under the Rational badge as Rational Application Developer) for implementation of web-services which is an important component of a service-oriented application. Finally, Section 7 contains concluding comments as part of the Discussion section.

2. Application Development Models

Application development models have evolved over the last few decades. Most of these changes in the application development models are a result of the technological evolutions that preceded them. For example, emergence of the personal computers made it possible to develop client/server applications. The major application development models include the monolithic host-centric model, client/server model, n-tiered model, and service-oriented model.

Monolithic host-centric model was dominant during the 1960s and 1970s. In this approach, monolithic applications were developed using programming languages such as COBOL and FORTRAN. These applications were developed mainly to run on mini- and mainframe computing platforms. All of the application processing was performed on the host computer. Mainframe-based monolithic applications are still being used in banking, airline, insurance, telecomm, and other industries as these applications support critical business processes. Although overall cost of mainframe applications is usually high, mainframe based applications' strengths include high availability and scalability. Thus, they are used in high-volume transaction processing environments.

Availability of PCs and networks led to the development of the client/server or 2-tiered application development model. One of the major driving forces for the popularity of this model was the enhanced graphical user interface that was made possible by the use of PCs as clients. In an application designed using this model, the database is usually deployed on an organizational server and the presentation is performed on client PCs. The rest of the application consisting mainly of the application logic can be split between the server and client PC in many different configurations. In a 'fat' client environment a large portion of the application logic is executed on the client PC, while in a 'thin' client environment most or all of the application logic resides on the server (Watt et al., 2002).

The client/server (2-tiered) model was extended to three-tiered model consisting of three distinct application components, namely, presentation, business logic, and data. These three components could reside on three different platforms. With the emergence of the Internet in the 1990s, the three-tiered model evolved into the n-tiered architecture used in distributed Web-enabled applications. Applications developed using this approach provide more-agile software architecture, and are easier and cost-effective to maintain compared to mainframe based applications (Mitchell, 2006). Many of the software development projects today use this software development model to develop Web-enabled distributed applications.

3. Service-oriented architecture (SOA)

In today's competitive business environment, ability to quickly adapt automated business processes in response to external and/or internal changes is a must. Many business applications that exist today in large organizations were developed within departmental boundaries and had functional instead of business-process focus. Such functional focus based application development resulted in existence of heterogeneous stovepipe applications that have interoperability issues. To automate business processes, functionalities embedded within these heterogeneous stovepipe departmental applications have to be weaved together. In the past, this was usually achieved by tightly integrating these applications using point-to-point solutions. Such integration is acceptable as long as the business processes are stable and integration of additional applications is required rarely.

In today's business environment, business processes are more likely to be dynamic and collaborative, making it necessary to adapt business processes more often, and integrate processes across organizational boundaries with business partners. Further, organizations also need to expose the business-critical functionalities embedded within the millions of lines of mainframe code developed over the past four decades (Mitchell, 2006), and integrate these functionalities with new business applications being developed using current technologies. Service-oriented architecture can enable organizations to develop and support dynamic, complex, and collaborative business processes that may span within and/or across organizational boundaries using existing heterogeneous application assets and new applications. It provides a cost-effective and efficient alternative to tight integration of heterogeneous applications to support organizational business processes (Janssen, et al., 2006). SOA has been effectively used by organizations to transform complicated heterogeneous information system infrastructure into seamless, streamlined, easy to maintain infrastructure, thus helping organizations in controlling their integration costs (Sanchez, 2006).

An SOA is an enterprise architecture that supports organizational business processes via the use of solutions conceived from the composition of distributed services, where a service is a well-defined, repeatable business task that can be performed by an application (Uleman, 2006). A *service* is usually implemented as a coarse-grained software unit that exists as a single instance, and interacts with other services and/or applications by using a loosely coupled, message based communication model (Brown et al., 2005).

SOA paradigm presents an approach in which modular, accessible, self-describing, implementation-independent, interoperable, and reusable components are published as services which can be remotely invoked and consumed by other applications or combined with other services (Fremantle et al., 2002; Stencil, 2002). *Thus at the heart of SOA is a collection of services that communicate with each other, and these services are used as building blocks of applications.* SOA provides an approach for how to describe and organize services to support their discovery and use. Each component in a SOA may perform one or more of the following three roles that are essential for services to be discovered and used: (1) service provider – publish the availability of services, (2) service broker – register and categorize published services and provide search capabilities, and (3) service requester – use service broker to find a service and use it to build an application (IBM, 2004).

Few of the characteristics that are associated with SOA include the following: (1) use of open standards based technologies, (2) the basic building blocks are coarse-grained services that can be published and are discoverable, (3) services are loosely coupled with other services to implement solutions to satisfy business needs, (4) reuse of existing software assets, and (5) higher level of abstraction in design (Uleman, 2006). Loose coupling between services make solutions developed using this approach to be more flexible and scalable compared to those developed using tight coupling between software components. This allows IT to react quickly to the changes in business requirements, which enhances an organization's ability to compete effectively and efficiently. Also, the higher level of abstraction used in this approach ensures that the focus is on the business tasks that constitute various business processes.

Finally, one of the distinctive features of SOA is the concept of top-down policy based approach of governance throughout the lifecycle of services at the enterprise level. This is necessary to effectively manage services within the enterprise in terms of functionality, interoperability, quality (service levels), security, regulatory compliance, reusability, maintainability, retirement, etc. Use of such governance structure allows IT to enforce uniform standards across the organization to introduce new services, and maintain and retire existing services. This governance structure permits creation of a library that lists available services, and will prevent creation of redundant services and improve reuse of services.

4. Web Services

W3C Web Services Architecture Group defines a Web service as a *software system identified by a universal resource identifier (URI), whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols* (W3C Web Services Architecture Group, 2004). Although it is possible to implement SOA without the use of Web services, Web services have been suggested to be the preferred approach to implementing SOA as its use significantly simplifies loose coupling between business components (Bloomberg, 2005; IBM, 2004).

The Web services framework at the minimum consists of a collection of three standards - standard to support communication between interacting services, standard to describe services, and standard to publish and discover services. The three main standards that enable implementation of Web services are the Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). These three standards are briefly discussed below:

- **SOAP** is an XML-based protocol to support communication between a Web service, its clients, and UDDI registry. It facilitates communication between Web services and interacting application components across heterogeneous platforms without requiring the use of custom binaries or other platform-specific information (Fremantle et al., 2002; Joshi et al., 2004). It provides a mechanism to invoke a Web service.

- **WSDL** is an XML-based standardized interface definition language used to describe what a Web service can do, where it resides, and how it can be invoked. A WSDL file associated with a Web service contains important details about the Web-service interface for client-service interaction. WSDL is used to provide definition of a Web service and interface specification for it.

- **UDDI** standard is used to publish, discover, and manage Web services in an UDDI registry. UDDI registry is like Yellow Pages that list available Web services that can be discovered by an application. Creation of a registry that contains available services allows service consumers to discover and invoke Web services that are published within the UDDI registry.

As shown in Figure 1, a service provider can publish a Web service and register it in a service registry. A Web services requestor, which can be an application running on any type of computing platform, can discover a Web service that has been published in the service registry. After discovering the Web service, the client interacts with it by sending and receiving SOAP messages. Each Web service entry published in the service registry points to a WSDL file that contains information for invoking and binding a Web service.

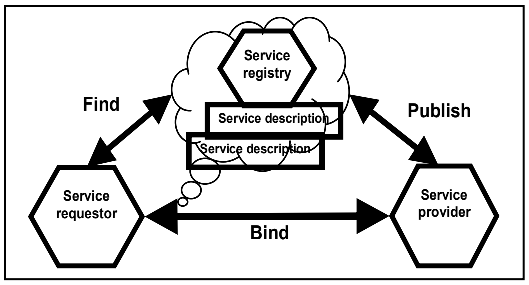

Figure 1 - Web Services Framework

5. Example of successful SOA implementation

While many companies have benefited from the use of SOA, Wachovia's implementation is perhaps one of the best illustrations of a successful SOA implementation. SOA has been deployed by many financial institutions to transform their complicated technology infrastructures into streamlined and easy to maintain technology infrastructure (Sanchez, 2006). Wachovia Bank's Corporate and Investment Banking division spent multi-hundred million dollars to revamp its existing software architecture to transform it into a service-oriented architecture (Margulius, 2006). They moved to software development strategy that revolved around a core of reusable frameworks, components, and services that can be leveraged by each of its business unit. The final outcome of this transformation process was an end-to-end, service-oriented development and delivery platform that was supported by a business-focused, product management culture within the information technology department. Some of the resulting benefits included, increased IT productivity due to reduction in development of similar software components by different groups within the bank, faster development of applications that end-users need due to the reuse of existing components, and above all, a standards-based application development framework. It has also allowed Wachovia Bank to win several multimillion dollar contracts, where the key selling point to the customer was the bank's superior technology architecture that permitted easy integration with customers' disparate systems (Margulius, 2006).

6. Implementation of service-oriented application using IBM WebSphere Studio Application Developer

To demonstrate the process of building a service-oriented application, we will demonstrate development of a simple inventory search application. The application will be designed and implemented using the Model-View-Controller (MVC) solution framework (see Figure 2). The MVC enforces the separation of presentation logic and business logic, and has been suggested as the blueprint that application developers should use to design and implement Web solutions (Takagiwa *et al.*, 2002). The entire application will be developed using IBM WebSphere Application developer Studio (WSAD) version 5.1.2. For the database environment, we will use an Oracle 9i database server.

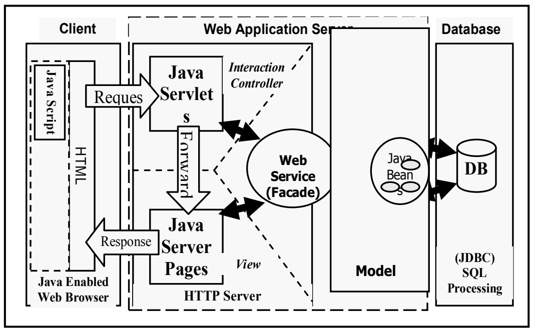

Figure 2 – Model-View-Controller Solution Framework

6.1 Model-View-Controller

Models are meant to serve as a computational abstraction of real entities. For example, a model of a product in an e-commerce application contains the identity and quantity of the inventories that are available for the product and offers functions to add and remove inventories. In J2EE (Java 2 Platform Enterprise Edition) space, the model components are typically written as Java Beans and represented by a Unified Modeling Language (UML) class diagram. In a service-oriented architecture, a Web service is created to transports the model components to the Web service clients.

Views present (render) the information that is contained in a model. For example, a graphical view may render a product with a list that shows the quantity and location of inventories, with a "remove" button next to each inventory. Views are rendered by a Web browser using HTML, which in J2EE space is typically produced by Java Server pages (JSPs).

A controller component controls the application flow. It accepts input from the user, interprets the input, and invokes the appropriate operation in the model in response to the input. For example, when the controller receives a request from the "remove" button of an item, it invokes the remove operation for that item to remove it from inventory. The controller then forwards the JSP view that displays the updated product without the item removed from inventory. In J2EE space, the controller is normally implemented as a Java Servlet.

6.2 An example of a service-oriented application

The inventory search application is a simple application that runs on the Web. The application is illustrated in Figure 3. The user enters search parameters for product identity and quantity fields, and then clicks the search button. The application responds by displaying a message to inform the user whether there is adequate inventory.

This application may be designed as a service-oriented application by implementing two Web projects: Web Service and Web Service client using WSAD; and it is developed using the following steps:

- The Web service project is first created in WSAD.
- Develop model components (Java Beans) to represent data returned by the Web service in the Web service project.
- Develop a facade bean that is a part of the model to define operations provided by the Web service. The facade is a J2EE design pattern to encapsulate business logic and business data and expose their interfaces, and thus the complexity the distributed services, to the clients (Alur *et al.*, 2001).
- In WSAD, a wizard is provided to generate the Web service and the Web service client project using the facade.
- Develop the JSP page and Servlet in the Web service client project.

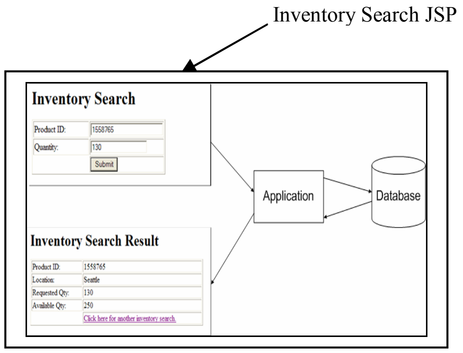

Figure 3 – Inventory Search Application

6.3 UML class diagram for representing model components

The class diagram illustrated in Figure 4 is created in a Web project using the IBM WSAD UML (Unified Modeling Language) visualization feature. The diagram consists of a few classes and associations. These are described as below:

- Product: a product of the system. There are many products.
- Inventory: a product inventory available in a warehouse location. A product may be inventoried at multiple warehouse locations.
- ProductAccess: a facade bean provides the service interfaces for finding all products or a product by a product identity number. This access bean executes queries against the database tables, populates product beans using the query results, and returns the product beans to the clients.

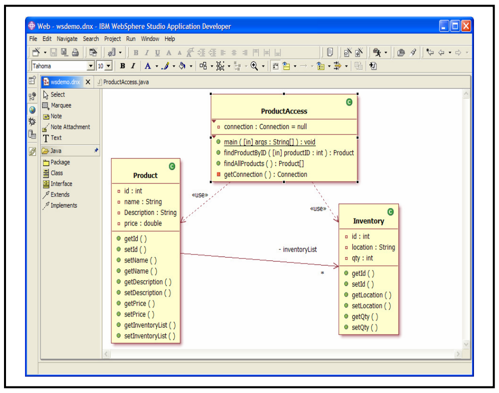

Figure 4 – Class Diagram

6.4 Generating Web service using a WebSphere wizard

IBM WSAD provides a wizard to generate a Web service from a Java Bean. We create a Web service using the facade bean (e.g. ProductAccess) that defines the Web service interface. After the wizard guides us through the Web service generation steps, the Web service is produced and a client project is created. The following files are included in the client project (see the Figure 5):

- WSDL document: this standard document uses XML to describe the Web service (what it does, interface it supports, etc.). The WSDL document also specifies the data (e.g. Product and Inventory) returned from the Web service and operations (e.g. findAllProducts and findProductByID) to access the Web service.
- Proxy bean: this bean runs in the client, and provides Java methods to call the remote operations on the Web service as if the methods were local.
- Test JSP: this JSP page runs in a Web browser and enables us to test the designed Web service. We select a method, enter the parameter, click the Invoke button, and inspect the result.

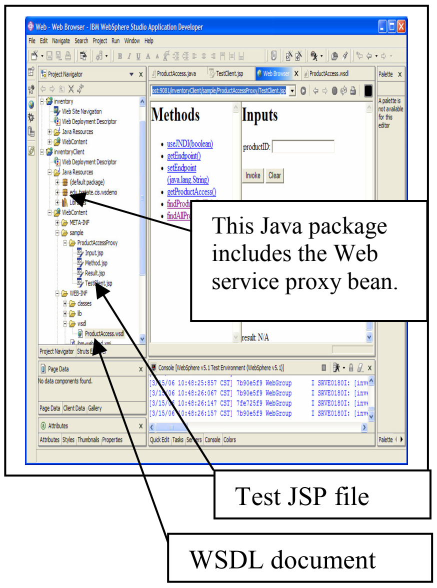

Figure 5 – Web Services Generation in WebSphere

6.5 Developing a Web service client

A Web service client project contains views and controllers. Using the Inventory Search JSP in Figure 3, this example demonstrates the views' common features of providing a user input form and displaying property values of Java Beans. The InventorySearchAction controller receives the input data, performs a Web service operation, and passes result Java Beans to the view. The details of the JSP and Servlet controller are described below.

After opening the JSP page, the user enters a product ID and a quantity to check the availability of the product. The page shows an out-of-stock message if no Product Java Bean is returned; otherwise, it receives a Product Java Bean and its Inventory Java Bean for which available quantity is greater than the input quantity. For the successful search, the page displays the product name and the warehouse location that maintains the adequate inventory.

The InventorySearchAction controller that is implemented by a Servlet is activated when the user submits the JSP form. First the controller receives the product ID and quantity. Second, it instantiates the Web service proxy bean, obtains a remote ProductAccess facade bean using the proxy, and calls the findProductByID method of the facade by the product ID (see the Figure 6). Third, if a product Java Bean were found to contain a list of Inventory Java Beans, the controller

```
//get parameters
int productID = new Integer(
req.getParameter("productID")).intValue();
int qty = new Inte-
ger(req.getParameter("qty")).intValue();

//instantiate a web service proxy
ProductAccessProxy proxy = new ProductAccessProxy();
ProductAccess access = proxy.getProductAccess();

//find a product
Product product = access.findProductByID(productID);

…
```

Figure 6 – Servlet Implementation

checks all of the list elements to see if there is an Inventory Java Bean with quantity greater than the input quantity. Finally, if both the Product and Inventory Java Beans were found, they are passed back to the JSP page.

7. Discussion

Use of SOA provides organizations with greater flexibility and control of the solutions they deploy (Brown et al., 2005). The foundation of SOA is the use of standards based approach for description, publication, discovery, selection, and binding of basic services (Papazoglou and Georgakopoulos, 2003). SOA promotes the idea of using loosely coupled services, each of which represent a block of business functionality, that can be assembled together to support a business process. Thus, SOA can be used by organizations to support business process driven application integration within and across organizational boundaries. This will enable organizations to quickly adapt their business processes in response to changes in business requirements, and also allow them to integrate their applications relatively easily with their business partners' applications.

When adopted at an enterprise level, SOA can provide an organization with a comprehensive plan regarding how IT can support business. However, it is important to note that SOA is an architecture, which is well reflected in the following note – *"Never forget that SOA is architecture – you can't buy it from a vendor, and you can't build it with programming code. Architecture is a set of best practices that guide your implementations, regardless of the technologies you choose to implement them"* (Bloomberg, 2005).

Many organizations have begun implementation of SOA and have reaped significant benefits. Implementation of SOA may require organizations to invest significant dollar amount, as indicated in Wachovia Bank's example (Margulius, 2006), and most likely will also necessitate a change in the organizational IT culture and practices (Brown et al., 2005).

8. References

Alur, D., Crupi, J., and Malks, D. (2001). "Core J2EE Patterns: Best Practices and Design Strategies." Palo Alto, California: Sun Microsystems Press.

Andriole, S.J. (2006). "The Collaborate/Integrate Business Technology Strategy." Communications of the ACM, Vol. 49, No. 5, pp. 85-90.

Bloomberg, J. (2005). "The SOA Pilot Pitfall." www.zapthink.com, available at: http://www.zapthink.com/report.html?id=ZAPFLASH-2005711

Brown, A.W., Delbaere, M., Eeles, P., Johnston, S., and Weaver, R. (2005). "Realizing Service-oriented Solutions with the IBM Rational Software Development Platform." IBM Systems Journal, Vol. 44, No. 4, pp. 727-752.

Fremantle, P., Weerawarana, S., and Khalaf, R. (2002). "Enterprise Services." Communications of the ACM, Vol. 45, No. 10, pp. 77-82.

IBM (2004). "Service-oriented Architectures and Web Services." IBM Developerworks Technical Presentation, Texas State University, Fall 2004.

Janssen, M., Gortmaker, J., and Wagenaar, R.W. (2006). "Web Service Orchestration in Public Administration: Challenges, Roles, and Growth Stages." Information Systems Management, Spring, Vol. 23, No. 2, pp. 44–55.

Joshi, P., Singh, H., and Phippen, A.D. (2004). "Web Services: Measuring Practitioner Attitude." Internet Research, Vol. 14, No. 5, pp. 366-371.

Margulius, D.L. (2006). "Banking on SOA." InfoWorld, July 13, Available at: http://www.infoworld.com/article/06/07/13/29FE wachovia_1.html

Mitchell, R.L. (2006). "Morphing the Mainframe." Computerworld, Vol. 30, No. 5, pp. 29-31.

Papazoglou, M.P. and Georgakopoulos (2003). "Services-oriented Computing." Communications of the ACM, Vol. 46, No. 10, pp. 25-28.

Sanchez, F. (2006). "The SOA Approach to Integration and Transformation." U. S. Banker, Vol. 116, July 2006, pp. 12-13.

Stencil (2002). "The Laws of Evolution: A Pragmatic Analysis of the Emerging Web Services Market." The Stencil Group: http://www.stencilgroup.com.

Takagiwa, O., Korchmar, J., Lindquist, A., and Vojtko, M. (2002). "WebSphere Studio Application Developer Programming Guide (1st ed.)." San Jose, California: IBM Corporation, International Technical Support Organization.

Uleman, R. (2006). "Service Oriented Architecture Unveiled." Geospatial Solutions, Vol. 16, No. 6, pp. 30-33.

Watt, E.R., Denoncourt, D., Lee, S., Stevens, R., and Cancilla, B. (2002). "Understanding e-Business Application Integration." Double Oak, Texas: MC Press, LLC.

W3C Web Services Architecture Group (2004). "Web Services Architecture Requirements." Editors: Austin, D., Barbir, A., Ferris, C. and Garg, S., Available at: http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/