The Business Value of the Service Component Architecture (SCA) and Service Data Objects (SDO)

A Whitepaper by IBM

Version 0.9

November 2005

Author

Rick Weaver
Senior Certified Consulting IT Specialist
Portfolio Manager: WebSphere Tools

Copyright Notice

I. Introduction

Most companies today are being pressured by their customers and shareholders to drive growth by improving productivity and limiting costs in every aspect of their operation. It is impossible to maximize efficiency if the company has rigid, expensive, proprietary IT systems. In fact, the most valuable thing a company can buy itself as an organization these days is **flexibility**, that is, flexibility to meet new market demands and seize opportunities before they are lost. By viewing its business as a collection of interconnected functions - discrete processes and services, such as check customer credit or authenticate user - and then deciding which of those functions are core or differentiating, and which can be streamlined or even outsourced, a company can achieve much greater flexibility. If the company can then mix and match these functions at will - or on the fly, in response to changing business conditions – the company will have tremendous competitive advantage in the marketplace. It is a powerful idea. But to achieve this degree of flexibility in the business operations, the company will need an equally flexible IT environment. It needs a service-oriented architecture, or SOA.

The need to respond to changing business demands with flexible IT solutions has led many businesses to Service-Oriented Architectures. SOA is a framework that combines individual business functions and processes, called services, to implement sophisticated business applications and processes. SOA is an approach to IT that considers business processes as reusable components or services which are **loosely-coupled** and that are platform and implementation **neutral**. The approach allows you to design solutions as assemblies of services in which the assembly description is a managed, **well-defined** first-class aspect of the solution, and hence, amenable to analysis, change, and evolution. The solution can then be viewed as a choreographed set of service interactions. The idea of viewing enterprise solutions as federations of services connected via well-specified contracts is gaining widespread support from customers and software vendors alike. The ultimate goal of adopting an SOA is to align the goals of the business with its IT assets in new and **standard** ways to making both more efficient at adapting to **change**.

Over the years, we've seen a variety of programming models such as CORBA, RMI, COM, and DCOM. These programming models facilitate direct invocations between tightly-coupled components. In most cases, these programming models only supported invocations between components within the enterprise.

These programming models rely on platform specific object models and implementation of objects and proxies using the same programming language. This is unrealistic as we often need to have multiple platforms 'working together' to provide business value. Web Services address part of this problem but Web services, in and of themselves, are insufficient to realize the value of SOA.

To fully realize the value of SOA, a technology independent programming model is needed to enable a flexible IT environment which can more easily align business objectives with solutions implemented by IT. A SOA programming model, following

SOA principles, should therefore be loosely-coupled itself. It should be independent of and abstracted from the underlying connectivity infrastructure, transport protocols, user interfaces, data access, pathways, and characteristics imposed on the service implementation by the application environment for which the service is hosted.

This paper will describe the capabilities in the SOA programming model and focus on the business value the SOA programming model will bring to you and your organization.

II. Introduction to the SOA Programming Model

A programming model can be defined as a pattern for building and deploying software applications and solutions. A programming model must define and represent the structure and operation of the software application. Specifically, the SOA programming model is used to define the structure and operation of composite applications, how they use services and how they aggregate these services together. It defines the key elements of the solution, how those elements are created, how those elements are linked together to form a cohesive solution, and how the solution is deployed into an appropriate operating environment. For SOA-based solutions, the programming model can be divided into the following elements:

Elements	Description	Technology used for Implementation
User Interaction	How a user interacts with a service, business process, or composite application	JavaServer Faces, Portlets, Rich Clients (including hand-held devices)
Invocation	How services are connected together and how services integrate and interoperate with each other.	Service Component Architecture (SCA), Enterprise Service Bus (ESB)
Composition	Composing services together builds a composite application. This can also include choreographing services to create an executable business process	Service Component Architecture (SCA), Business Process Execution Language (WS-BPEL)
Business Components	Relevant units of business logic built as components with interfaces that are independent of the underlying implementation details	Service Component Architecture (SCA)
Information	A uniform way of representing data	Service Data Objects (SDO)

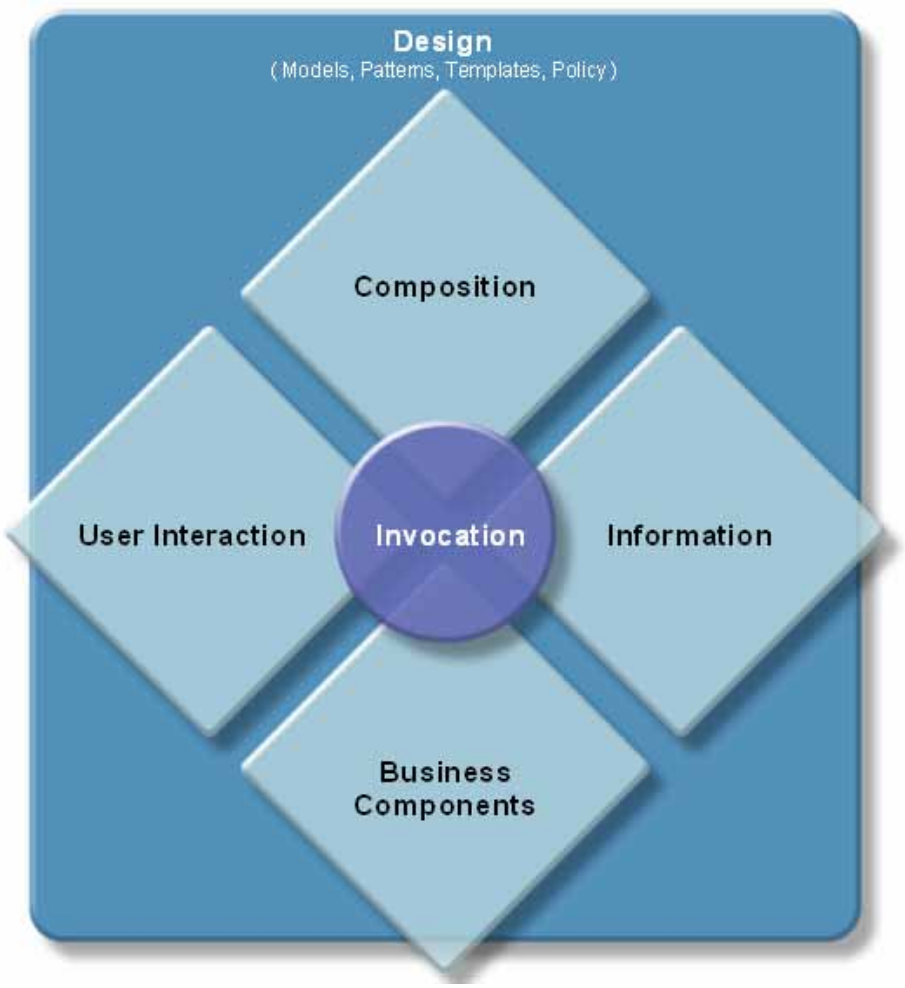

Figure 1 – A Conceptual View of the SOA Programming Model

Technology such as Java Server Faces, Portlets, WS-BPEL, and the concept of an enterprise services bus have been around for some time. IBM has a wide-array of products which implement these technologies as evidenced by: WebSphere Application Server, WebSphere Message Broker, WebSphere MQ, WebSphere Portal Server, WebSphere Process Server, and WebSphere Enterprise Service Bus.

Service Component Architecture (SCA) and Service Data Objects (SDO) represent new technology that allows solutions to be built based on the SOA programming model. Together, they are designed to provide the ability to separate the SOA solution into its appropriate elements and to simplify the representation of service-oriented business logic and associated data.

The Service Component Architecture provides the ability to represent business logic as a reusable component which can be easily integrated when assembling an application or solution. We often refer to the assembled application as a *Composite Application*.

SCA imposes a strict separation between the concerns of defining a component implementation from that of usage of that service. Through this separation we achieve

platform and programming language neutrality in the composite applications which are built using SCA. A developer can focus on assembling composite applications and business goals rather than on the underlying implementation and infrastructure technology (Java, J2EE, C++, Web Services, JMS, JCA, etc).

Service Data Objects are designed to simplify data access and how data is represented in a SOA solution. SDOs provide a consistent and uniform way to access data versus the diverse data access models such as JDBC Result Sets, JCA Records, DOM, JAXB, and EJB entities. SDOs can be used to provide a uniform way of creating, retrieving, updating, and deleting business data regardless of how the data is physically accessed, providing for both static and dynamic programming styles as well as connected and disconnected styles of access. SDOs make developers more productive by freeing them from the technical details of how to access a particular back-end data source.

SCA and SDO are used together in the SOA programming model. Business Components are represented as SCA components and the data used between components are represented as SDOs.

III. Business Value of SCA and SDO

The primary value of SCA/SDO is in aiding the development of solutions by focusing on business objectives instead of the technical details as to how that business objective is implemented. This benefits everyone, from a company using SCA and SDO to 3rd party vendors building components for consumption.

The value that SCA and SCO bring to the business also brings these benefits to IT:

- **Improved Flexibility**: Ease of making changes to business processes with minimal impact on other parts of the system.
- **Increased Programmer Productivity:** Using a single unified programming model that encapsulates technical implementation details will accelerate the development and deployment of composite applications.

A. Improved Flexibility

Flexibility is one of the key benefits of the SOA programming model. From a business point of view, flexibility can have several meanings:

- ***Technology Neutral.*** The business is able to substitute implementations or change protocols, deployment targets or other environmental concerns without changing the SCA application.

- ***Reuse.*** Clearly defined, loosely-coupled services are the hallmark of any well implemented SOA, and as such, lend themselves for easy inclusion in other business processes and composite applications.

- ***Composition.*** Services can be composed together through SCA assemblies to build more complex composite applications.

- ***Adaptability to change.*** Allowing business analysts to substitute new pricing models without having to 'wait' overnight or for a weekend maintenance run. Note: This does not imply you would bypass testing! In many cases, you would still want to test a new pricing model before deploying into production.

To illustrate business flexibility, we will consider an example of a travel booking system (illustrated in Figure 2):

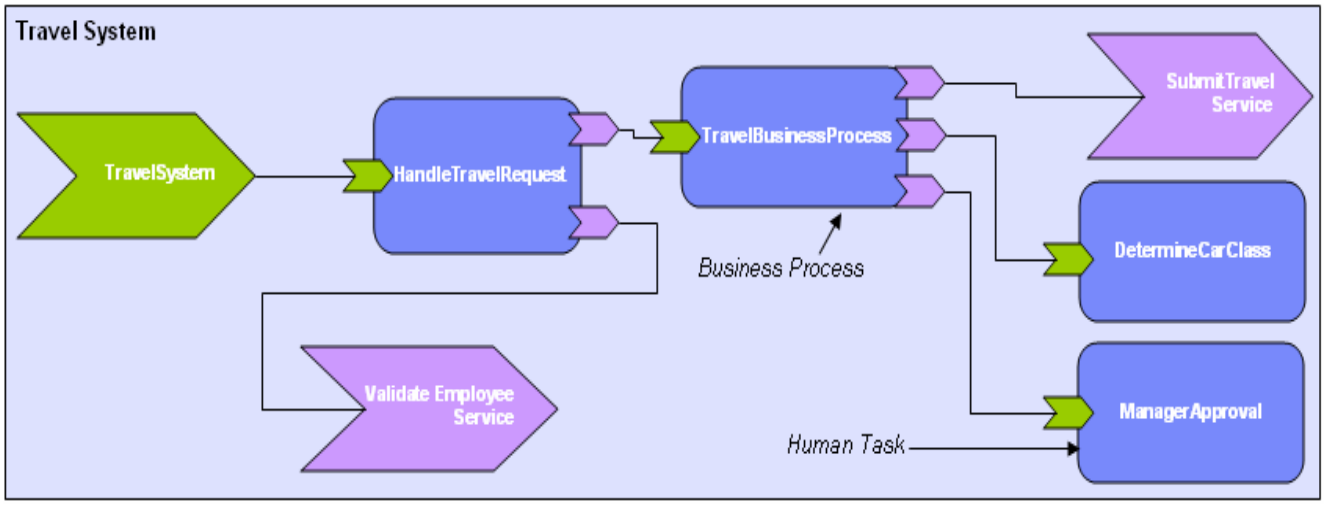

Figure 2: An Assembly View of a Travel Booking System

The assembly view in Figure 2 illustrates various components and the services they offer. It also shows how components reference services offered by other components. The services offered in a component are defined via an interface. This interface represents the contract between components. This view of components and their referenced services is central to how solutions are assembled using SCA.

The rectangular boxes in Figure 2 represent business components (e.g. HandleTravelRequest, TravelBusinessProcess, etc) that make up a travel system. The TravelBusinessProcess component has been implemented as a business process (built using WS-BPEL). The TravelBusinessProcess contains references to other components that it uses. One component it references is a ManagerApproval component. This component currently represents a human task in which the manager of the employee must approve the travel request.

To illustrate the agility of the SOA application model, let's examine a scenario which involves re-composition of this application. Suppose that following the successful deployment of the travel system, an issue of business agility has been identified with our business process – managers are taking too long to approve or reject a travel request. In

lieu of sending reminder emails to the managers, the composite application is transformed by replacing the ManagerApproval human task component with an automated business component (perhaps an EJB or a business rule to approve or reject the travel request).

The only modification we need to make is to reassemble the travel booking system to utilize the new service (illustrated in Figure 3).

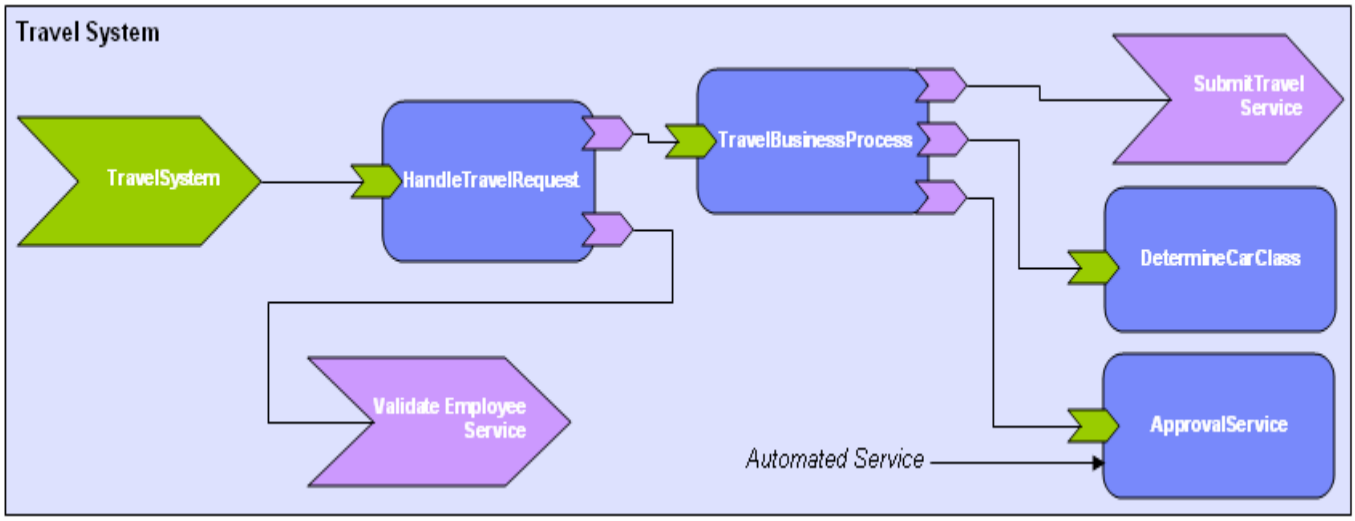

Figure 3: Travel Booking System rewired to use an automated service

Notice that the TravelBusinessProcess isn't modified at all. There is no need to modify any component that depends on the Approval service. We simply rewire the solution to use the new ApprovalService component.

In this example, the ManagerApproval and ApprovalService have the same interface so we could easily substitute one service with another by simply rewiring.

What if the new service we are substituting has a different interface? We might introduce another component which mediates the request and response between the two components (As illustrated in figure 4). Even through the ApprovalService2 has a different interface than the original ManagerApproval component; the mediation component can map the interface differences so that nothing else in the composite application needs to be modified.

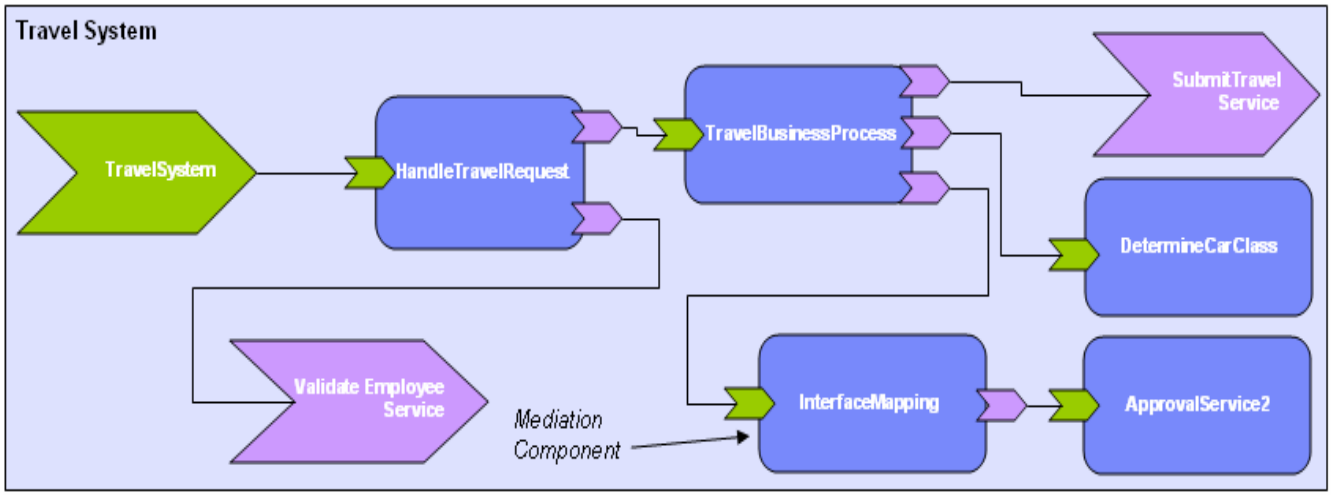

Figure 4: Adding a mediation component to map between interfaces

Other Examples of Flexibility

Let's look at other examples of how we can use the SOA programming model to inject flexibility into the composite applications we develop:

- Example 1: We can use *service data objects* (SDO) to inject flexibility at the data layer. The objective in using SDOs to add flexibility is to avoid rewriting components to keep up with shifting technology used to access data. For example, consider a Java component which offers a service which reads flight schedules from a database. Today, the component uses JDBC. Tomorrow, the topology changes and the data is provided through a web service. Using SDOs, the implementation for the component accessing the data does not need to change. If SDOs were not used, substantial rework would be needed to substitute the Web service application programming interface (API) in the Java component.

- Example 2: Service Component Architecture provides the capability of defining "qualities of service" such as transactions, security and reliable asynchronous invocation declaratively. This allows for greater flexibility when applying or changing a quality of service without requiring detailed programming.

B. Increased Programmer Productivity

The other benefit of the SOA programming model we'll focus on is improved programmer productivity. The same qualities of SCA programming that provide the business with flexibility also provides the business with greater productivity. Programmer productivity provides value in many ways but the **business** value of programmer productivity is in increasing the speed in which we bring IT solutions to bear on business problems with fewer defects. The SOA programming model accomplishes this by:

- ***Separation of concerns.*** The loosely coupled service model with clearly defined service definitions enables the SOA development team to work in parallel and independently of each other.

- ***Service Reuse.*** Not only does reuse increase the business's flexibility, it also reduces the cost of building new SOA applications. We can use SCA to uniformly represent legacy assets as well as newly engineered components using the same fundamental principles. Assembling legacy assets into a composite application allows the support of a bottom-up style of development.

- ***Top-Down Development:*** In addition to a bottom-style of development, the SOA Programming model also allows for the development of composite applications to be done in a top-down manner. Components can be assembled before the component is actually implemented! The actual component implementation choices can occur later in the development cycle.

- ***Improved Organization.*** SCA introduces the notion of modules. A module is used to group components together as we build a composite application. Having thousands of services that aren't group together logically would be unwieldy without having a level of composition.

- ***Technology Neutral.*** SCA and SDO provide an abstraction which is intended to hide the individual complexities of the technologies that are used to connect the service consumer and service provider.

Separation of Concerns

The SOA programming model clearly separates the tasks of application assembly and application development. Figure 7 lists some of the typical roles in SOA development.

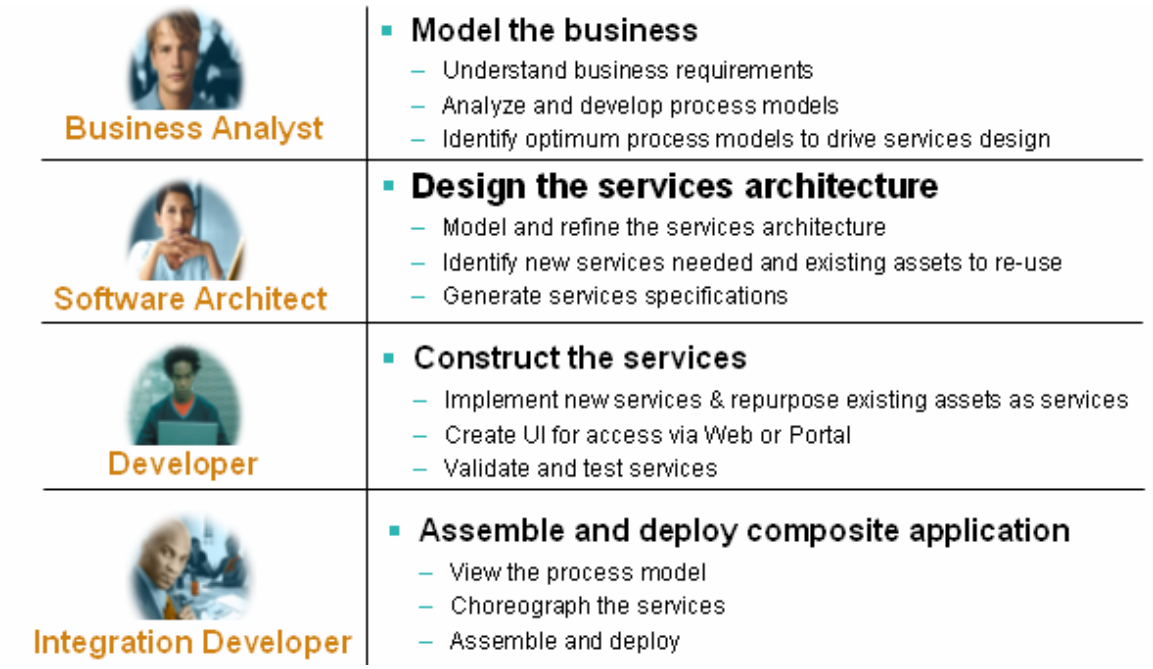

Figure 5: Key Roles in Service-Oriented Design and Development

The roles listed in Figure 5 are logical roles. Depending on organizational structure, it might be common for a single person to span roles.

At first glance, one might think that having four roles in SOA development seems complex. In reality, this actually helps simplify the development process by separation of concerns between these roles.

To illustrate this, you'll notice that Figure 7 lists two types of developers:

- A **Developer** who implements new services and new components
- An **Integration Developer** who assembles components together to form a composite application.

The Integration Developer does not need to be concerned with how a component is implemented (e.g. Java, .NET, J2EE, COBOL, C++, etc). The Integration Developer is primarily interested in *what* a component does and what its runtime capabilities are, not *how* a component does it. This allows an Integration Developer to more rapidly assemble services together to form a composite application.

Reducing the technical skills

One of the goals of a SOA programming model is the ability to easily re-compose an application. In order to achieve real business value, it must be so easy that it is not necessary to involve a highly skilled developer.

To illustrate this, let's look at a simplified travel booking scenario implemented as a J2EE Application.

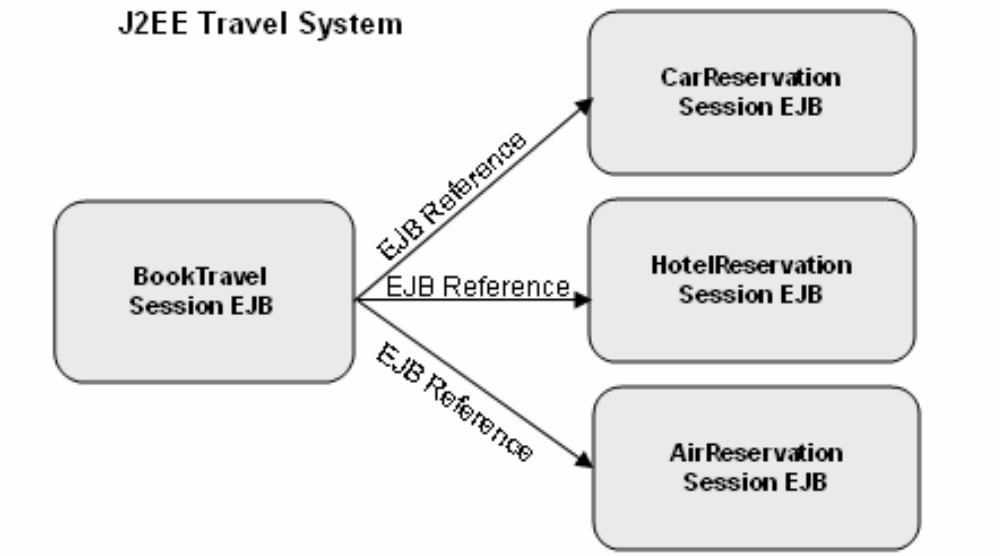

Figure 6: A Travel Booking System implemented as a J2EE application.

The BookTravel session EJB serves as the primary component in the travel booking system and has dependencies on other EJBs (CarReservation, HotelReservation, and AirReservation).

The J2EE developer implementing BookTravel would use standard J2EE programming to use the CarReservation EJB:

```
// Get the initial context as shown in a previous example
...
// Look up the home interface using the JNDI name
try {
   java.lang.Object ejbHome =
      initialContext.lookup(
         "java:comp/env/com/ejb/CarReservation");
   carHome =
(CarReservationHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, CarReservationHome.class);
}
   catch (NamingException e) { // Error getting the home interface
   ...
```

The code shown above illustrates looking up one EJB using the specific J2EE APIs, namely the Java Naming and Directory Interface (JNDI). Similar code would be duplicated to lookup the HotelReservation and AirReservation EJBs.

What happens if we need to change the travel booking system due to changing service level agreements? What if we need to substitute a new component for AirReservation?

If the new AirReservation component was another EJB component, the code change would be relatively minor for a Java developer *but would still require coding changes*. The changes the Java developer would need to make would involve:

1. Modifying the lookup for the new AirReservation component (or rebind the ejb reference).
2. Modifying code in order to handle any differences in the data.

What if the new AirReservation was a non-J2EE artifact such as a web service? In the J2EE travel booking system, the impact would be **far** more significant.

1. The Java developer would need to replace logic to access the EJB with logic to access the Web Service.
2. They would be introducing two programming models into their BookTravel system. (EJB and Web Services)
3. The Java developer needs skills for both programming models
4. There is little flexibility in the system. Any future changes to the travel booking system will involve programming changes.

Let's now look at the same travel booking system using the SOA programming model.

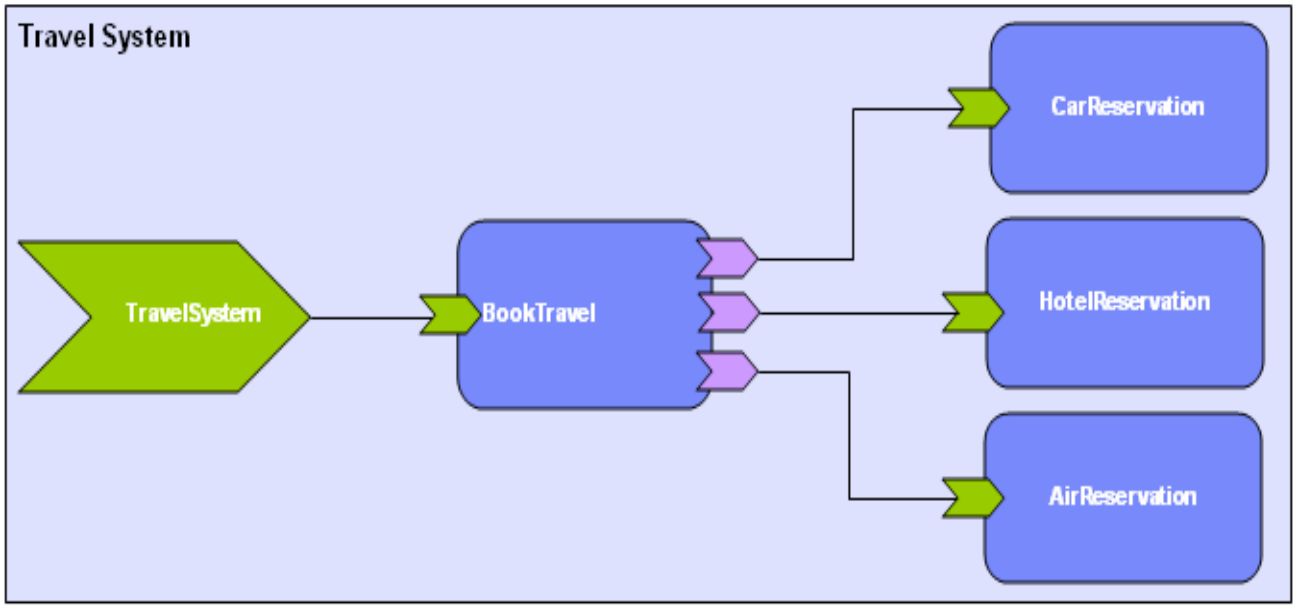

Figure 7: A Travel Booking System assembled using the SOA programming model.

Let's contrast this with making a change to the travel booking system built using the SOA programming model. Regardless of whether the change involves replacing AirReservation with a J2EE or Web service component:

1. An Integration Developer (not the Java Developer) makes the change.
2. The Integration Developer deals with only one programming model.

3. The new DetermineAir component is added to the application assembly editor.
4. No code is changed.
5. The system is more flexible when change occurs.

A key observation to make about the person using the SOA programming model does not need the same level of technical skills to make the change. By contrast, deeper and more specialized technical skills will be required to handle changes in the J2EE based application.

Even if a Java developer using SCA were dealing with the change from an EJB to a Web Service, their code would remain exactly the same:

```
ModuleContext moduleContext=CurrentModuleContext.getContext();
AirReservation airReservation=
  (AirReservation)moduleContext.locateService(“AirReservation”);
airReservation.bookFlight(flight);
```

IV. Summary

Businesses must be flexible to handle the many challenges that exist. Increased pressures from customers and competitors require business agility which in turn requires IT agility. As illustrated in this paper, adapting the SOA programming model can help improve business agility. A key business value of adapting the SOA programming model is to allow IT to build composite applications by focusing on business objectives, business goals, and business processes instead of the technical details as to how this is implemented.

As with the proper design of any system, IT should make proper architecture decisions regarding how the SOA programming model is applied. Applying the SOA programming model to every fine grained interaction between business logic components could result in an overly-complicated, under performing application. A key to success will be through the thoughtful application of the SOA programming model to inject flexibility into coarse grained, loosely coupled services of a composite application. As illustrated in Figure 5, the role of the Software Architect plays an important role in helping to make the proper architectural decisions as they model and refine the services architecture.

The end result will be an increased business value of IT in the business solutions they provide.

V. References

[1] SCA Specification
http://www.ibm.com/developerworks/library/specification/ws-sca/

[2] SDO Specification
http://www.ibm.com/developerworks/library/specification/ws-sdo/

[3] SCA Sample application “Building your first application – Simplified BigBank”
http://www.ibm.com/developerworks/library/specification/ws-sca/

[4] SCA White Paper (technical)
http://www.ibm.com/developerworks/library/specification/ws-sca/

VI. Acknowledgements

For their assistance in preparing this paper, I would like to thank Graham Barber, Alan Brown, Michael Beisiegel, Dave Booz, Mike Edwards, Bill Hassell, Eric Herness, Stephen Kinder and Kareem Yusuf.