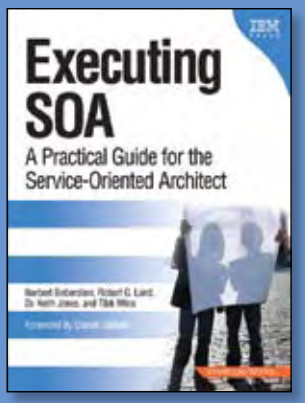
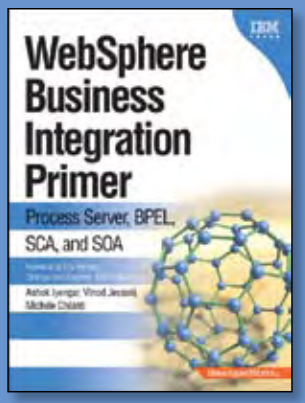
# Summer SOA and WebSphere Sampler

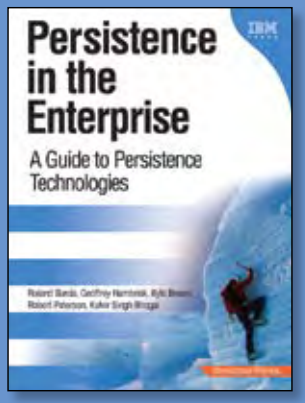
SEE INSIDE FOR **35% OFF OFFER**

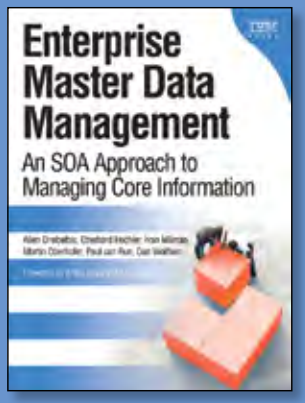
Excerpts from Recently Published and Classic Books from IBM Press






IBM Press™

COMPLIMENTS OF IBM PRESS

ibmpressbooks.com

# SPECTACULAR SUMMER SPECIALS

**35% DISCOUNT AND FREE SHIPPING**

Once you've sampled our 6 recently published and classic SOA and WebSphere books featured here, you can buy them at a **35% discount** at **ibmpressbooks/SOAspecial**. Enter the code **SOASPECIAL** at checkout to receive the **35% discount**.

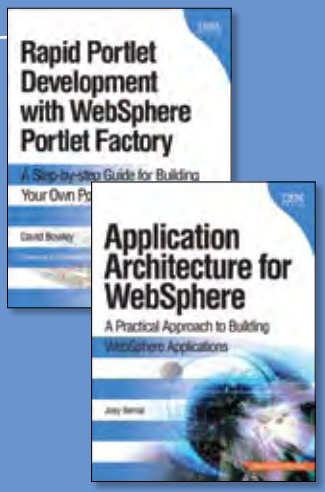
Plus we offer **FREE** FedEx Ground Shipping in the US. If you're ordering outside of the US, see our **UK bookstore** for their special offer.


Plus, as an added bonus, when you place an order for two books — one of these 6 featured books plus any other IBM Press book — you'll receive a **FREE** copy of ***The New Language of Business: SOA and Web 2.0.***

**DON'T WAIT** — this offer is only available while supplies last. Enter the code **SOASPECIAL** at checkout. Qualifying orders (one of the 6 featured SOA and WebSphere books, plus any second IBM Press book) will receive *The New Language of Business* in a separate shipment — please allow 3 weeks for arrival.

## Stay Tuned for 2 Upcoming Books

We've included the table of contents for 2 new books publishing this fall. Sign up for the monthly IBM Press newsletter **ibmpress.com/newsletters/index.asp** for updates on these books' publication and to receive an introductory offer upon their publication.



# Summer SOA and WebSphere Sampler

eBook

**Executing SOA — 0132353741**

Norbert Bieberstein | Robert G. Laird | Dr. Keith Jones | Tilak Mitra

**CHAPTER 4: A METHODOLOGY FOR SERVICE MODELING AND DESIGN**

**WebSphere Business Integration Primer — 013224831X**

Ashok Iyengar | Vinod Jessani | Michele Chilanti

**CHAPTER 2: BUSINESS INTEGRATION ARCHITECTURE AND PATTERNS**

**The New Language of Business — 013195654X**

Sandy Carter

**CHAPTER 11: PUTTING IT ALL TOGETHER**

**Service-Oriented Architecture (SOA) Compass — 0131870025**

Norbert Bieberstein | Sanjay Bose | Marc Fiammante | Dr. Keith Jones | Rawn Shah

**CHAPTER 4: SOA PROJECT PLANNING ASPECTS**

**Persistence in the Enterprise — 0131587560**

Roland Barcia | Geoffrey Hambrick | Kyle Brown | Robert Peterson | Kulvir Singh Bhogal

**CHAPTER 4: EVALUATING YOUR OPTIONS**

**Enterprise Master Data Management — 0132366258**

Allen Dreibelbis | Eberhard Hechler | Ivan Milman

Martin Oberhofer | Paul van Run | Dan Wolfson

**CHAPTER 2: MDM AS AN SOA ENABLER**

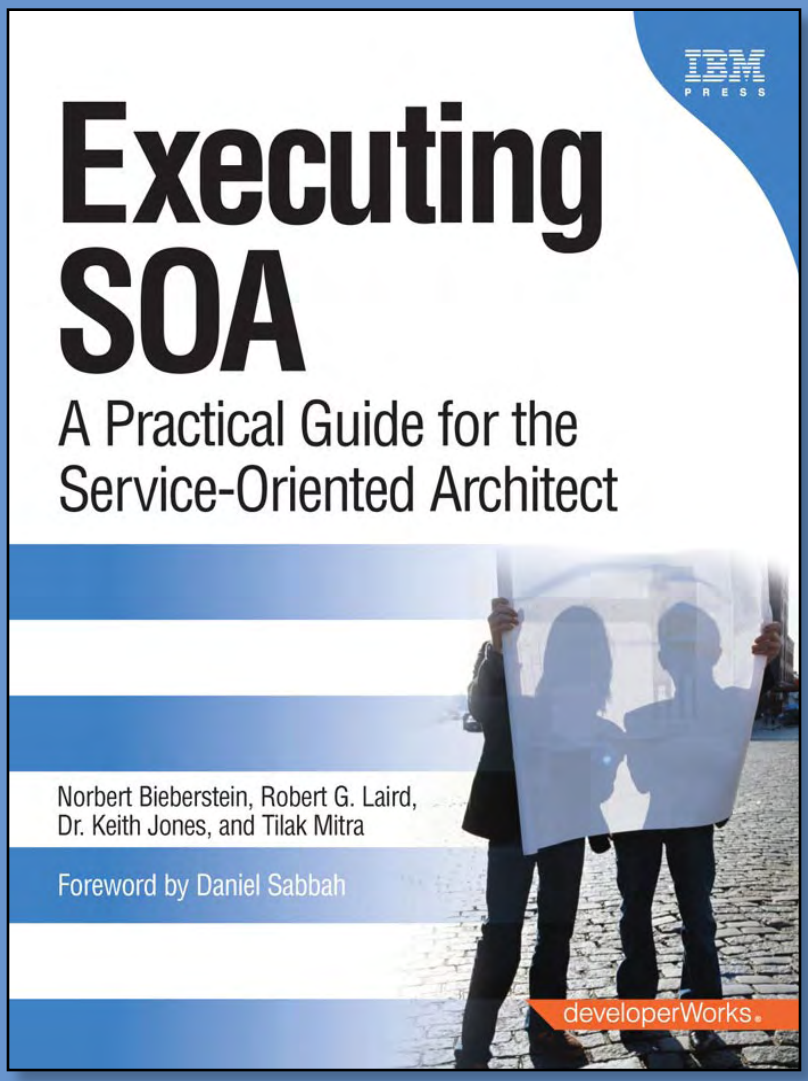
**Upcoming SOA and WebSphere Books**

**COMING FALL 2008**

**Safari Books Online**

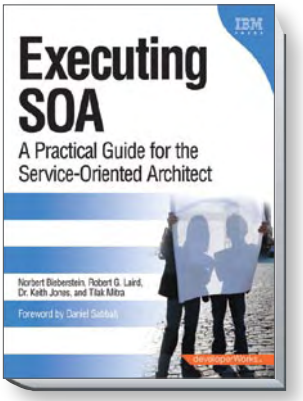
**MORE INFORMATION**

Published by:
Pearson Education
800 East 96th Street
Indianapolis, IN 46240 USA



Norbert Bieberstein
Robert G. Laird
Dr. Keith Jones
Tilak Mitra






# Executing SOA

## A Practical Guide for the Service-Oriented Architect

### Table of Contents



 ISBN: 0-13-235374-1

### About the Authors

**NORBERT BIEBERSTEIN,** an IBM solution architect, is responsible for communicating progress with SOA offerings. In total, he has more than 27 years of experience in IT and computer sciences.

**ROBERT G. LAIRD** is an IT architect with IBM in the SOA Advanced Technologies group, performing worldwide consulting for IBM customers in the area of SOA governance and SOA architecture since May, 2006.

**DR. KEITH JONES** is currently an executive IT architect with IBM in the SOA Advanced Technologies team where he focuses on the definition and implementation of service-oriented architectures with leading-edge customers. He has 30 years experience in the IT industry.

**TILAK MITRA,** senior certified executive IT architect at IBM Global Services, specializes in helping IBM customers to conceptualize, envision, develop a roadmap, design, and implement enterprise architectures based on SOA.

IBM Press™

Chapter 4

# A Methodology for Service Modeling and Design

When the programming model shifted from the traditional procedural model to that of object-orientation, a major paradigm shift occurred in the world of IT development. The focus was on encapsulating the state and behavior of entities and calling that encapsulation a class. Instances of a class were called objects, which occupied some space in the memory. Object orientation (OO) brought in concepts of inheritance, encapsulation, and polymorphism that could be applied to define relationships between classes. With the prevalence of the use of OO in the programming world, developers and architects started noticing some patterns that can be applied to the usage of OO principles to solve similar types of problems. The patterns depicted the deconstruction of a problem into multiple class entities, together with their interrelationships using the basic concepts of OO, to provide a solution to the problem. The seminal work in this field was done by the Gang of Four authors in the book called *Design Patterns: Elements of Reusable Object-Oriented Software*. (See the "References" section.) Whereas in OO the first-class constructs were objects and classes, the next-generation methodology for building software applications was called component-based development (CBD). In CBD, the first-class constructs were components, where a component was defined by its external specification, which could be used without any knowledge of its internal implementation. As such, the same external specification could be implemented in different programming language (for example, Java, C#). The internal implementation of a component may use multiple classes that collectively provide the implementation of the external specification. The classes could use one or more design patterns, thereby leveraging the advantages of OO principles.

In SOA, the main emphasis is on the identification of the right services followed by their specification and realization. Although some might argue that object-oriented analysis and design (OOAD) techniques can be used as a good starting point for services, its main emphasis is on microlevel abstractions. Services, on the other hand, are business-aligned entities and therefore are at a much higher level of abstraction than are objects and components.

The main first-class constructs in an SOA are *services, service components,* and *process flows*. For the sake of brevity, we refer to process flows as just flows. These are at a level of abstraction that is higher than that of objects, classes, and components. Hence, there needs to be a higher level of modeling and design principles that deal with the first-class constructs of an SOA. Service-oriented modeling and design is a discipline that provides prescriptive guidance about how to effectively design an SOA using services, service components, and flows. Rational Software, now a part of IBM, has provided an extension to Rational Unified Process (RUP) called *RUP-SOMA* (see the "References" section), which is built on a service-oriented analysis and design technique developed by IBM called Service Oriented Modeling and Architecture (SOMA). The rest of this chapter takes you through the SOMA technique and explains how it helps in the identification, specification, and realization of services, service components, and flows.

## 4.1 An SOA Reference Architecture


A.4.1

When defining a service-oriented solution, it makes sense to keep a reference architecture in context—an architecture that establishes the building blocks of SOA: *services, service components,* and *flows* that collectively support enterprise business processes and the business goals. The reference architecture provides characteristics and definitions for each layer and the relationships between them and assists in the placement of the architectural building blocks onto each layer. This layering facilitates the creation of architectural blueprints in SOA and helps in reusability of solutions and assets within an industry and potentially across industry verticals. Figure 4-1 shows a sample logical SOA reference architecture.

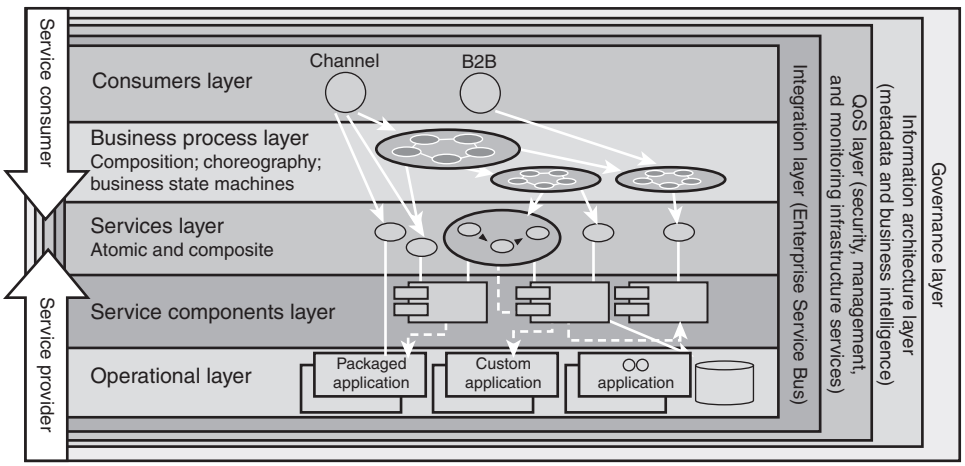



**Figure 4-1** Logical view of SOA Reference Architecture

The figure shows a nine-layered architecture with five horizontal layers and four vertical layers. The horizontal layers follow the basic principle of a layered architecture model in which

architecture building blocks (ABB) from layers above can access ABBs from layers below, whereas layers below may not access ABBs from layers above. The vertical layers usually contain ABBs that are cross-cutting in nature, which implies that they may be applicable to and used by ABBs in one or more of the horizontal layers. This can also be called a partial layered architecture because any layer above does not need to strictly interact with elements from its immediate lower layer. For example, a specific access channel can directly access a service rather than needing to go through a business process. The access constraints, however, are dictated by the architectural style, guidelines, and principles that apply to a given SOA solution. This view of the SOA reference architecture is independent of any specific technology implementation, and hence is a logical view. Instances of this logical architecture can be developed for a specific platform and technology. Following are definitions of each of the layers:

- **Layer 1: Operational systems**—This layer includes the operational systems that exist in the current IT environment of the enterprise, supporting business activities. Operational systems include all custom applications, packaged applications, legacy systems, transaction-processing systems, and the various databases.
- **Layer 2: Service component layer**—Components in this layer conform to the contracts defined by services in the services layer. A service component may realize one or more services. A service component provides an implementation façade that aggregates functionality from multiple, possible disparate, operational systems while hiding the integration and access complexities from the service that is exposed to the consumer. The consumer thus is oblivious of the service component, which encapsulates the implementation complexities. The advantage of this façade component comes from the flexibility of changing operational systems without affecting the service definition. The service component provides an enforcement point for service realization to ensure quality of service (QoS) and compliance to service level agreements.
- **Layer 3: Services layer**—This layer include all the services defined in the enterprise service portfolio. The definition of each service, which constitutes both its syntactic and semantic information, is defined in this layer. Whereas the syntactic information is essentially around the operations on each service, the input and output messages, and the definition of the service faults, the semantic information is around the service policies, service management decisions, service access requirements, and so on. The services are defined in such a way that they are accessible to and invocable by channels and consumers independent of implementation and the transport protocol. The critical step is the identification of the services using the various techniques that can be employed for the same. The methodology that we focus on in this chapter addresses such identification techniques.
- **Layer 4: Business process layer**—Business processes depict how the business runs. A business process is an IT representation of the various activities coordinated and collaborated in an enterprise to perform a specific high-level business function. This layer represents the processes as an orchestration or a composition of loosely coupled services—leveraging the services represented in the services layer. The layer is also responsible for the entire lifecycle management of the processes along with

their orchestration, and choreography. The data and information flow between steps within each process is also represented in this layer. Processes represented in this layer are the connection medium between business requirements and their manifestation as IT-level solutions using ABBs from other horizontal and vertical layers in the architecture stack. Users, channels, and B2B partner systems in the consumer layer uses the business processes in this layer as one of the ways to invoke application functionality.

- **Layer 5: Consumer layer**—This layer depicts the various channels through which the IT functions are delivered. The channels can be in the form of different user types (for example, external and internal consumers who access application functionality through access mechanisms like B2B systems, portals, rich clients, and other forms). The goal of this layer is to standardize on the access protocol and data format to enable the quick creation of front ends to the business processes and services exposed from the layers below. Some such standards have emerged in the form of portlets, service component architecture (SCA) components, and Web Services for Remote Portlets (WSRP). The adherence to standard mechanisms for developing the presentation layer components for the business processes and services helps in providing template solutions in the form of standard architecture patterns, which helps the developer community to adopt common front-end patterns for service consumption.
- **Layer 6: Integration layer**—This layer provides the capability for service consumers to locate service providers and initiate service invocations. Through the three basic capabilities of mediation, routing, and data and protocol transformation, this layer helps foster a service ecosystem wherein services can communicate with each other while being a part of a business process. The key nonfunctional requirements such as security, latency, and quality of service between adjacent layers in the reference architecture are implemented by the architecture building blocks in this layer. The functions of this layer are typically and increasingly being collectively defined as the enterprise service bus (ESB). An ESB is a collection of architecture patterns that uses open standards and protocols to implement the three basic capabilities of this layer and provide a layer of indirection between the service consumers and the service provider by exposing the services only through the ESB. ESB products usually add some specialized features to provide differentiated capabilities in the marketplace.

  The integration capabilities are most commonly used by ABBs residing between Layer 2 through Layer 5. As an example, in Layer 5 there can be many consumers accessing enterprise services through different channel types. Each channel type can use different protocols—HTML, WML (for mobile users), and Voice XML (for IVR users), to name a few. Each of these protocols and message formats may be passed through an Extensible Stylesheet Language Transformations (XSLT) engine before the actual service is invoked. This XSLT transform is usually an ESB-provided feature. The beauty of the ESB-based integration layer is that any feature or function that can be exposed in a manner that follows open standards and protocols for access can be plugged into the ESB so that it is enabled to take part in a service-based ecosystem. Figure 4-2 depicts a logical view of the ESB.

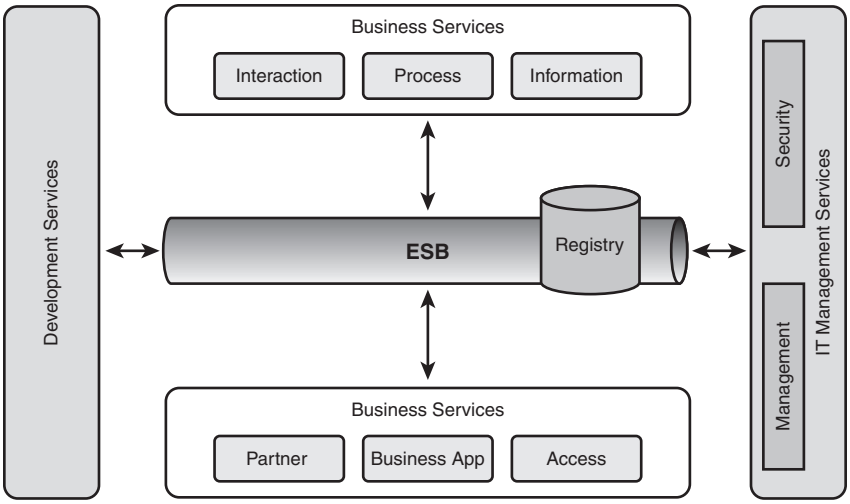



**Figure 4-2** Logical view of an ESB in the integration layer

As Figure 4-2 suggests, the ESB provides capabilities for service consumers and providers to connect to each other, for services to be discovered using the registry, for services to be managed and for secure invocations, and provides application programming interfaces (API) to facilitate the development of service connectivity.

- **Layer 7: QoS layer**—This layer focuses on implementing and managing the nonfunctional requirements (NFR) that the services need to implement. Although SOA brings some real value proposition through the new architectural style, the programming model that supports the building of the first-class SOA constructs adds some inherent challenges that are nontrivial to address. The challenges arise while trying to comply with the essential tenets of SOA: abstraction, open standards and protocols, distributed computing, heterogeneous computing infrastructures, federated service ecosystem, and so on. Adherence to these compliance requirements often makes the implementation of the NFRs that much more complicated. This layer provides the infrastructure capabilities to realize the NFRs. It captures the data elements that provide the information around noncompliance to NFRs at each of the horizontal layers. Standard NFRs that it monitors for noncompliance is security, availability, scalability, and reliability.
- **Layer 8: Information architecture layer**—This layer ensures a proper representation of the data and information that is required in an SOA. The data architecture and the information architecture representation (along with its key considerations and guidelines for its design and usage) at each specific horizontal layer are the responsibilities of this layer.

  Industry models (for example, ACORD, IAA, JXDD) and their usage to define the information architecture, along with business protocols used to exchange business

data, are addressed in this layer. It also stores the metadata required for data mining and business intelligence. Refer to the "References" section for the details of some industry models.

- **Layer 9: Governance layer**—This layer ensures the proper management of the entire lifecycle of the services. It is responsible for prioritizing which high-value services should be implemented, for each of the layers in the architecture, and for providing a rationalization based on how the service satisfies a business or IT goal of the enterprise. Enforcing both design-time and runtime policies that the services should implement and conform to is one of the key responsibilities of this layer. Essentially, this layer provides a framework that efficiently oversees the design and implementation of services so that they comply with the various business and IT regulatory policies and requirements.
  Chapter 3, "SOA Governance," discusses SOA governance and the responsibilities of a governance council in detail. Those responsibilities all feature in this layer of the reference architecture.

It is worth noting that one of the primary reasons for the SOA solution stack representation is that it helps to communicate, to the business and IT stakeholders, the evolution and realization of the enterprises SOA vision and roadmap through iterative implementation. Communicating with the stakeholders is key to ensure that the business commitment is pervasive across the various phases of an SOA initiative.

The methodology that we discuss in this chapter will help in identifying, specifying, and realizing the first-class constructs of an SOA and their placement in the various layers of the architecture stack. This logical view of the SOA reference architecture is also known as the SOA solution stack or just the solution stack. Therefore, the terms *SOA reference architecture, SOA solution stack,* and *solution stack* all refer to the same concept and hence can be used interchangeably.

## 4.2 Service Oriented Modeling and Architecture

A.4.2

Service Oriented Modeling and Architecture (SOMA) is a modeling and design technique developed by IBM that provides prescriptive steps for how to enable target business processes by defining and developing a service-based IT solution. SOMA provides the communication link between the business requirements and the IT solution. It provides guidance on how to use business model and information as inputs to derive and define a service-based IT model. SOMA, as a methodology, addresses the gap between SOA and object orientation. This methodology approach provides modeling, analysis, design techniques, and activities to define the foundations of an SOA. It helps defining the elements in each of the SOA layers (see Figure 4-2) and also to take architectural decisions at each level.

At the heart of SOMA is the identification and specification of services, components, and process flows. At a high level, SOMA is a three-phased approach to *identify, specify,* and *realize* services, components, and flows (typically, choreography of services). The first phase is that of service identification, where various techniques are used to identify an exhaustive

list of candidate services. The second phase is that of service specification, in which a detailed design of services and components is completed. The realization phase focuses on making architectural decisions, justifying the most prudent approach to implement the services.

SOMA focuses directly on the services, service components, and flows. These SOA constructs reside between Layer 2 and Layer 4 of the architecture stack. However, the activities performed as a part of the end-to-end methodology influence the placement of components in the other layers of the stack. Figure 4-3 illustrates the focus of SOMA as it pertains to the solution stack.

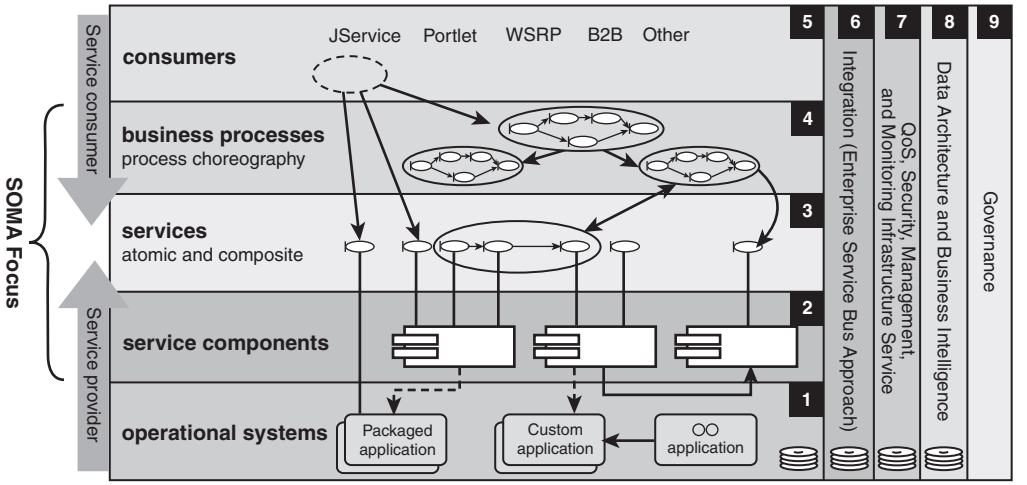



**Figure 4-3** The focus of SOMA is on Layer 2 through Layer 4.

One of the main outputs of the SOMA method is a service model. It is recommended that a service model constitute of the following artifacts about services:

- **Service portfolio**—List of all the enterprise services
- **Service hierarchy**—A categorization of services into service groups
- **Service exposure**—An analysis and rationalization of which services should be exposed and which should not
- **Service dependencies**—Representing the dependencies of services on other services
- **Service composition**—How services take part in compositions to realize business process flows
- **Service NFRs**—The nonfunctional requirements that a service must comply with
- **State management**—The various types of states that a service should maintain and implement
- **Realization decisions**—Architectural decisions, for each service, around the most justified mechanism to implement the service

The techniques employed in the various phases of the method address one or more parts of the service model. Although there may be other work products that can be developed by exercising the various phases and activities of this method, we concentrate on the service model in this chapter.

### 4.2.1 Validation of Inputs

Before we can start identifying, specifying, and realizing service, it is imperative to validate the inputs that the method expects. SOMA takes inputs from the business.

The "to-be" business processes—their definitions and their corresponding process models that are decomposed down to second and third levels—are mandatory inputs. Services that will be summoned for implementation will be used in an orchestration to choreograph business process as a network of collaborating services at runtime.

Acknowledging the fact that SOA is primarily a business initiative where we strive to achieve flexibility, responsiveness, and agility in the business, the emphasis is on using SOA principles to solve business problems by designing and implementing an IT solution aligned with the business goals and strategy. The business goals and drivers of the company, the realization of which is the basis for commissioning an IT project, is a very important input into the method. The business goals need to be supplemented with a mechanism to measure the success of achieving the goal. Key Performance Indicator (KPI) is a metric that provides the business a measure of success of a software service against the attainment criteria for a business goal. Hence, business goals, drivers, and their associated KPIs are very important inputs to the method. These KPIs are used to measure how effective and successful an enterprise SOA initiative has been.

SOA strongly recommends the concept of reuse. Therefore, the traditional and often scary "rip and replace" approach to systems development is the last option in the philosophical premise of SOA. The idea is to reuse as much of the functionality available in currently running enterprise systems as possible. A good and solid understanding of the current IT environment represents a vital input to the method. For instance, the applications and systems, the functionalities they provide, the importance and usage of the functionalities provided, and the roadmap for enhancement or sunset of each of the systems are all key inputs that help in gaining a good understanding of the current IT portfolio.

The current organizational design and the future organization scope and requirements can also prove to be invaluable inputs. These can be used to identify which line of business will be responsible for the ownership and funding of the service lifecycle. However, this is not a mandatory input and can be categorized as "nice to have information."

We recommend the execution of a method-adoption workshop for any client engagement. Such a workshop allows the consulting team to customize the method to suit the specific requirements of the client. In an SOA-based engagement, and as a part of this method-adoption workshop, one can determine the specific inputs available for use with the SOMA method. The inputs that are available must be carefully validated for completeness. So, what happens if they are incomplete?

The first thing to do is to assess the gaps between the required and the available information, and then a gap-mitigation plan needs to be put in place. A part of the recommended approach is to perform further interview sessions with the stakeholders and subject matter experts (SME) and use that information gathered therein to incorporate the missing information. We also recommend documenting customer pain points and use them to define the customer requirements, business drivers, and priorities. These are by no means the only two ways to address gaps between available and required inputs, and the IT team might have its own gap-mitigation plan as it suits the current customer scenario.

If the mandatory inputs are not available, however, a commitment needs to be made by the business stakeholders to make them available to the degree of completeness as requested by the IT team.

### 4.2.2 Identification

When the validation of inputs has been completed, the focus shifts to the identification of the services that will ultimately constitute the service portfolio. The aim is to get an exhaustive list of services that are potential candidates for exposure and then categorize the services into some logical grouping. Experience suggests that the general approaches taken to identify services are sometimes too restrictive; typically we do not exploit all possible sources to identify enterprise-level services. To come up with such an exhaustive list of "candidate" services, it is recommended to use a combination of three complementary techniques: domain decomposition, existing asset analysis, and goal service modeling. After we have compiled the list of candidate services, we use a technique that extracts, from the list of candidate services, only those services relevant for exposure. Figure 4-4 illustrates the three techniques for service identification.

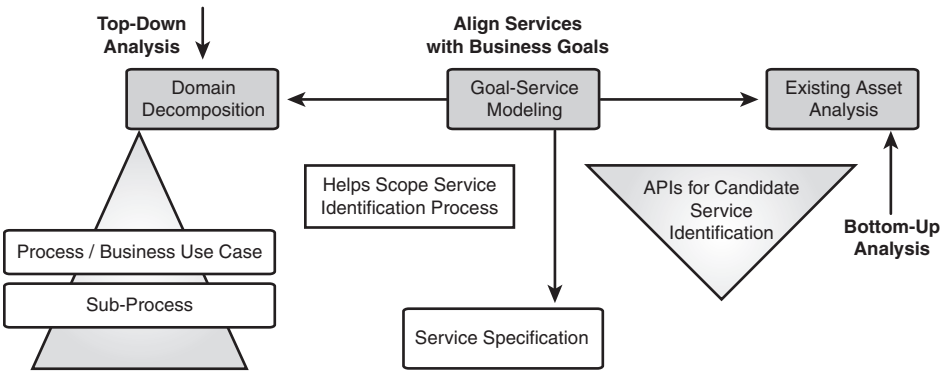



**Figure 4-4** The three different techniques for service identification

Let's explore further the various techniques for service identification.

#### 4.2.1.1 Domain Decomposition

This is a top-down technique that involves the decomposition of the business domain into its functional areas and subsystems, including the decomposition of its business processes into subprocesses and high-level business use cases. These use cases are often good candidates for business services exposed at the edge of the enterprise, or for those used within the boundaries of the enterprise across lines of business. Apart from identifying candidate services, this technique helps to identify functional areas that identify boundaries for subsystems.

This technique includes one step called functional area analysis (FAA). In FAA, we decompose the business domains into logical cohesive functional units and name each unit as a functional area. The resultant set of functional areas provides a modular view of the business and forms the basis of IT subsystem identification, nomenclature, and design. It is not necessary for FAA to be done as a part of SOMA because it can leverage similar work that could have been done as a part of any other project initiative in the same enterprise. The identification of functional areas assists in their usage in service categorization, wherein the identified candidate services can be categorized using the functional areas. The "Service Hierarchy" section of the service model work product is a formalization of the categorization of services.

FAA usually falls under the expertise realm of business analysts and domain experts. One can start with a base set of functional areas, but if and when services are identified and start to be grouped using the functional areas, one can refactor the existing functional areas so that they make sense from a service grouping standpoint. The key point we are trying to make here is that functional areas can be refactored to suit the proper grouping to services.

The next step in this technique is called process decomposition. In process decomposition, we decompose business processes into its constituent subprocesses and further into more atomic activities or tasks. The resultant process model depicts both the business-level and IT-level flow of events that realize a business process. It also forms the basis of candidate service identification. A process is a group of logically related activities that use the resources of the organization to provide defined results in support of the organization's objectives. Process models describe the work that an organization is involved in and the behavior of systems the organization uses. Each business process in the scope of the business or IT transformation is decomposed into subprocesses and further into leaf-level subprocesses. Each activity in the resultant process model or process breakdown tree is considered a candidate for service exposure. Hence, each is added to a list called the service portfolio. At this point, the service portfolio consists of all the subprocesses, activities, and tasks from the process model definitions for every single process. The "Service Portfolio" section of the service model work product is the recipient of the list of candidate services that are identified in this step.

Decomposition of processes into its activities and tasks also assists in identifying commonalities and variations between multiple business processes. The common activities or subprocesses provide good candidates for services while the points of variability enable the design of the system in a way that it fosters design resiliency and makes the system more adaptive to incorporate future changes. Variations in a system are usually identified across

three aspects: structures, processes, and rules. Externalizing these variability points enables configurable injection of flexibility into system design. Variations may also suggest new services based on types, processes, and rules.

#### 4.2.1.2 Existing Asset Analysis

Existing asset analysis is a bottom-up approach in which we examine assets such as existing custom applications, packaged applications and industry models to determine what can be leveraged to realize service functionality. This analysis is also designed to uncover any services that may have been missed through process decomposition. While you are analyzing existing legacy and custom applications, we recommend performing a coarse-grained mapping in which you map business functionality in the portfolio of existing applications to the business processes and determine which step (as identified through domain decomposition in Section 4.2.1.1) in the process can be potentially realized by some functionality in existing applications. We do not recommend performing a fine-grained mapping to specific transactions and batch processes within legacy application at this stage.

During the coarse-grained mapping activity, a detailed understanding of the application's state and quality is obtained that will allow the assessment of technical risks associated with the services that are going to be realized by the existing system functionality. For the applications that have such technical risks associated with their usage for service implementation, we recommend scoping some technical prototypes to test things like basic connectivity, protocol issues, data structures and formats, and so on. This prototyping will help mitigate the project risks that might otherwise crop up during the later stages, for example, during implementation.

So, with this technique, we can not only start thinking about service realizations using existing assets but also identify new services. These new services will be added to the service portfolio. At this point, the service portfolio consists of potential services derived from both a top-down and a bottom-up approach.

#### 4.2.1.3 Goal Service Modeling

Goal service modeling (GSM) is the third of the three techniques and is used to validate and unearth other services not captured by either top-down or bottom-up service identification approaches. It ensures that key services have not been missed. GSM provides the key link between the business goals and IT through the traceability of services directly to a business goal. The attainment of the goal, through the supporting service, is measured through the KPIs and its metrics that were documented as a part of the inputs from the business. GSM also ensures that stakeholder involvement and accountability is maintained through their consent on the business goals that needs to be achieved. Services directly linked to the business goals would then have a higher probability of being prioritized and funded for subsequent design and implementation. It is worthwhile to point out that GSM may be used as a scoping mechanism that assists in defining the scope of a project by focusing deeper into the problem domain. A problem domain is often too large to be tackled in one iteration and hence narrowing down and identifying an area that provides the highest impact (by realizing one or more business goals) to the business in a reasonable and acceptable timeframe is

a recommended way of scoping a project initiative. Once the scope is defined around a business goal, not only can services be identified through the GSM technique but also the top-down (domain decomposition) and bottom-up (existing asset analysis) techniques may be performed on the given scope of the project.

Identifying business goals is a nontrivial task. It is not uncommon for clients to be grappling for ways to articulate their real business goals. SOA architects are not the ideal consultants who can be of much help to the business people. The business analysts and SMEs are the ones who come to the rescue, helping the clients to clearly articulate their business goals.

The business goals are usually stated in a way that are too lofty and at a very high level. It is difficult and often impossible to try to identify and associate a service to these lofty goals. The recommended approach is to work closely with the business and domain SMEs to decompose the goals into subgoals, and keep decomposing until the point that a subgoal is actionable. *Actionable* here means the attainment of what I call as the "Aha!" factor—that I can identify an IT function that I can use to realize this subgoal. Hence, each business goal is usually decomposed into subgoals, and then services are identified that can realize them. This approach differs radically from the top-down and bottom-up techniques, and therefore you have a high potential of discovering new services. These new services are added back to the service portfolio. Some of the discovered services can be found to be already present in the existing service portfolio. This is a good thing, a validation step that ascertains that more than one technique for service identification has identified the same service!

So, what do we achieve in the service identification phase?

- We have taken a three-pronged approach to identify candidate services.
- Each identified candidate service is added to the service portfolio.
- FAA is performed or leveraged to provide a mechanism to group services—the service hierarchy.
- The service grouping may be iteratively refactored to provide the best categorization of services.
- For functionality in existing applications identified for use to realize service implementations, a technical assessment is performed to assess the viability of reusing the existing application for service implementation.

From a service model standpoint, what have we addressed?

- We are able to provide a service portfolio of candidate services.
- We categorized the services into a service hierarchy or grouping.

With this, we move on to the second phase in the method: specification of services.

### 4.2.3 Specification

The specification phase helps design the details of the three first-class constructs of SOA: services, service components, and flows. It uses a combination of three high-level activities to determine which services to expose, provides a detailed specification for the exposed services, and specifies the flows (processes) and service components. The three activities are

called service specification, subsystem analysis, and component specification. From a service model work product standpoint, this phase provides the most content: The service exposure, service dependencies, service composition, service NFRs, service messages, and state management are all addressed in this phase. The rest of this section focuses on the three activities.

#### 4.2.3.1 Service Specification

Service specification defines the dependencies, composition, exposure decisions, messages, QoS constraints, and decisions regarding the management of state within a service.

The first task concerns service exposure. The service portfolio had an exhaustive list of services obtained through the three techniques that we used for service identification. It is easy to comprehend that this list may contain too many candidate services; not all of them are at the right level of granularity to be exposed as services. Some of the service candidates may be too coarse grained and might actually be more like business processes or subprocesses rather than individual services (for example, some of the process elements derived from the first level of process decomposition), whereas some others may be too fine-grained IT functions (for example, the process elements in the lowest level of process decomposition and some of the existing system functionality). Deciding to expose the entire list of candidate services is a perfect recipe for following a perfect antipattern in SOA—the service proliferation syndrome (a phenomenon we want to avoid). Some economic and practical considerations limit the exposure of all candidate services. A cost is associated with every service chosen for exposure. The funding of the entire service lifecycle, the governance factor around service lifecycle management, and the added underlying infrastructure requirements to support security, scalability, performance, and other nonfunctional requirements make it impractical to follow the rules of economies of scale when it comes to exposing all candidate services.

Based on these premises, we recommend a service litmus test. The test consists of specific criteria applied to the candidate services. Only those services that meet the criteria are chosen for service exposure. The method provides an initial set of test criteria in the following form:

1. **Business alignment**—A service must be business aligned. If a service is not, in some shape or form, traceable back to a business goal, it may not be an ideal candidate to be chosen for exposure.
2. **Composability**—Tests the ability of a service to be used in a context entirely different from the one from which the service was originally identified. A service should be able to participate in multiple business processes without compromising the NFR compliance requirements for the process.
3. **Feasibility of implementation**—Tests the technical feasibility of implementing the service in a cost- and time-effective manner. Practical considerations limit overly complex services to be commissioned for implementation.
4. **Redundancy elimination**—Tests whether the service can be used within all processes and applications where its function is required.

This method by no means enforces these four litmus tests and recommends defining litmus test criteria taking the client environment, goals, and other relevant and pertinent client-specific factors into account. The most important point here is that some form of an elimination criterion needs to be defined that will allow the prioritization of services for exposure consideration. It is also recommended to provide a justification for the services that failed the litmus test for exposure. This justification, when documented, provides crucial information which becomes vital when the failed services might be revisited later in the scope of another SOA initiative in the same enterprise. It is quite possible that the business goals, client prerogatives, and other factors as applicable during subsequent projects might expose a service that might not have passed the exposure tests during the current scope!

Now that a filtered list of services has been determined, the services chosen for exposure constitute the refined service portfolio. Each service in this portfolio now needs to be provided with detailed specification. The first specification activity is to identify service dependencies.

Services are ideally dependent on other exposed services. Although in an ideal world of SOA everything is a service, in reality we find that services are more frequently dependent on underlying IT components. This dependency happens typically in situations where QoS requirements such as performance and availability tend to push SOA architects to design a service to be hardwired to one more technology-specific component. A service-to-service dependency is called a processing dependency because a service is dependent on one or more services only in the context of a given business process. Sometimes however, strict nonfunctional requirements mandate that services are more tightly coupled in their dependency on finer grained IT components. This type of dependency is often categorized as a functional dependency. Categorizing service dependencies into processing and functional groupings provides key architectural and design considerations for service implementation. Figure 4-5 provides an example of the two types of dependencies.

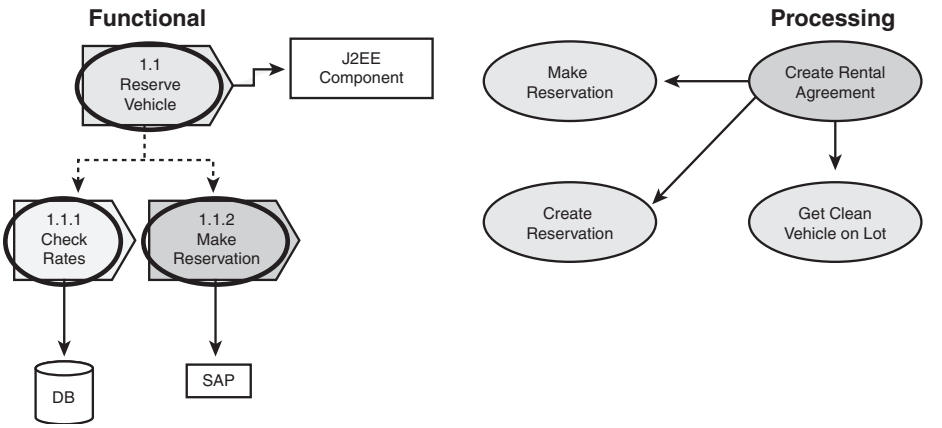



**Figure 4-5** The two general types of service dependencies

With service dependencies depicted, the next natural activity is to identify service composition and flows. Services that take part in a composition are a collection of related services to solve a specific high-level business function. A business process can be either represented as a single composite service or may be realized as an orchestration of one or more composites or individual services. Each business process that is in the scope of the transformation initiative is modeled as such an orchestration. These orchestrated services and their specifications and design will influence how they may be implemented as Business Process Execution Language (BPEL) flows at runtime. See the "References" section for more information on BPEL.

The next activity is identification of the NFRs that a service must implement. Each service must, in addition to the business logic that it implements, comply with a set of NFRs. The security mandates for service access, for example, authentication requirements for external consumers or no security for internal consumers; the availability of the service, whether 99.999 or 99.9; the maximum allowable latency for service-invocation turnaround, whether 1 millisecond or 1 minute are examples of NFRs that typically must be implemented by a given service. The method prescribes the documentation of all the required and optional NFRs for each service. This information will influence downstream microdesign and implementation. Note that keeping the complexity of business logic equal, the complexity of the NFRs of a service directly affects the time, cost, and resource requirements for its implementation.

Service message and its specification is one of the most critical and significant activities in this phase. A service message—the input message, the output message, and the exception and faults—typically constitutes the syntactic specification of the service. Service messages are usually defined in XML format for the obvious reasons of portability and interoperability. This method provides some prescriptive guidance for designing service messages.

One of the main tenets of SOA is to provide business flexibility and agility to an enterprise through an IT infrastructure that facilitates the enterprise to participate in a collaborative ecosystem. Collaboration brings in the key requirement for flexible and seamless integration with other collaborating entities. One of the first things you want to do is to standardize on the message format used to define services. Following a standard message format can facilitate a better integration with other partners outside the enterprise perimeter. A growing number of industry-specific consortiums provide standard definitions for business entities and information applicable to a given industry. For example, the insurance industry and the retail industry might define a *customer business entity* differently. The attributes and even some base and common operations on the entities are being standardized per industry. These standard specifications are called industry models. There exist quite a few stable industry models, such as ACORD for insurance, enhanced Telecommunications Operations Map (eTOM) for electronics, Justice XML Data Dictionary (JXDD) for the Department of Justice, Open Travel Alliance (OTA) for travel and transportation, and so on. Refer to the "References" section for more information on eTOM and OTA.

This method recommends using the industry model, if available for the given industry, as a starting point for message specification. Acknowledging that these specifications are often all encompassing, the first level of analysis that needs to be done is to define a subset of the

industry model artifacts that is applicable to the client wherein the SOA project is being undertaken. This subset of the industry model can be the base specifications. In more cases than not, there will be a need to add some specific extensions to the base specifications that incorporates client-specific requirements. The base specifications together with the extensions constitute what we call the Enterprise Message Format (EMF). Defining the EMF is the first step toward service interoperability. Sometimes, a version or a flavor of an EMF may already be present with the client. If so, it needs to be analyzed, validated, and enhanced to support the new and upcoming requirements. The input and output message elements must be compliant with the EMF. The EMF is also a perfect starting point to define the Enterprise Information Model (EIM) and it also influences the Logical Data Model (LDM), both of which, although are not necessarily a part of SOA, are mandatory architectural constructs in any enterprise application development.

**NOTE**

Note that the domain of information architecture plays a very important role in SOA. The data translation requirements together with information architecture that models the data and information flow from the consumer, through all the layers of the architecture stack right down to the operational systems layer falls under the domain of the Integration and Data architecture layers in the SOA reference architecture.

The amount of work that goes into the definition of the EMF and subsequently into service message specifications is often underestimated and becomes the widest chasm to bridge. Keep in mind that service message specification is closely, if not tightly, linked with the design of the information model and the data models and therefore not a trivial work effort.

Get your EMF well designed and documented; it will have a positive impact on downstream design and specification activities.

The last major activity focuses on analysis of the state management requirements for a service and its operations. As a general design rule, the business logic implemented by a service should not include state-specific logic.

However, requirements often mandate that some services address some state requirements. The most commonly occurring types of state are transactional state, functional state, and security state. Transaction state is required to support transactions spanning multiple messages. If an atomic transaction must include the results of multiple business actions preformed as a result of multiple messages from a service requestor to a service provider, the state of the transaction must be maintained until all the messages involved in the transaction are completed and the transaction is committed or rolled back.

Security state addresses how the identity of a consumer may be verified. In a stateless scenario, the client is authenticated each time a message is received. In a stateful scenario, a token is typically passed in the message sent by the consumer.

Functional state refers to state that must be maintained between messages while a business action is accomplished.

We must account for how the specific state requirements must be managed. Sometimes, IT technology components influence how state can be managed. For example, a security state can be managed by a product such as IBM Tivoli Access Manager (see the "References" section of Chapter 6 for more information), and the transactional state requirements between multiple service invocations in a business process may be managed by a BPEL engine. Hence, the documentation of the specific state requirements for each service is imperative. Keep in mind that during service realization, we can come up with the right architectural decisions justifying the best mechanism to implement the state management requirements. So, document them here!

This is just the first major activity during the specification phase; we have two more areas to address. Before we move on, however, we want to mention that all six recommended activities that we discussed as a part of this technique are iterative in nature and we will, depending on the scope and complexity of the project, need to run through this technique multiple times until we get a refined and robust specification of services at this level.

#### 4.2.3.2 Subsystem Analysis

Just like functional areas provide a logical grouping of a business domain, an IT subsystem is a semantically meaningful grouping of logically cohesive IT artifacts, which are in the form of components and classes. When a functional area is too large to grasp, it is broken down into these logical units called subsystems. Although this decomposition can be done top down or bottom up, the method recommends a top-down approach.

A subsystem consists of three types of components: service components, functional components, and technical components. It is the job of the SOA architect to identify a logical grouping of services that can be implemented together by a team that has specific domain knowledge in the area. Subsystem analysis and identification is nothing special to SOA, and it has been practiced from the days of OO; therefore, I call it "architecture as usual" (AAU) work. Let's focus now on the constituents of a subsystem and how to identify them.

A functional component is an IT component that encapsulates and supplies a single type of business functionality. For example, any customer-related business logic and IT APIs can be encapsulated in a single functional component called, for example, CustomerManager.

A technical component is an IT component that provides generic functionality such as authentication, error handling, and auditing. Their implementation is usually more closely tied with the technology platform that is used.

A service component is an IT component built as a coarse-grained façade on top of more focused and finer-grained functional and technical components. Think of it as a central component in a mediator pattern. A service is usually aligned to a high-level business function. For this business function to be implemented, it might need to call finer-grained IT APIs on some functional and technical components. Let's consider an example in context. Suppose a service is to provide "details of the last reserved vehicle for a customer." This requirement will typically necessitate a call to a CustomerManager component to "retrieve the customer profile" information, use some business logic to pick the relevant customer details from the returned result, and then invoke a VehicleManager component to "retrieve

the current reserved vehicle" for the given customer. In the process, it may invoke a technical component called AuditManager to log the service request. Neither the CustomerManager nor the VehicleManager nor the AuditManager has the business logic to control this microflow of steps. This control of the microflow, which component to invoke and in which sequence to realize the service request, is the responsibility of the service component. The service component can be designed to conform to the specifications of service component architecture (SCA). See Chapter 6, "Realization of Services," for detailed treatment of SCA.

For illustrative purposes only, Figure 4-6 depicts how subsystems are related to service, functional, and technical components.

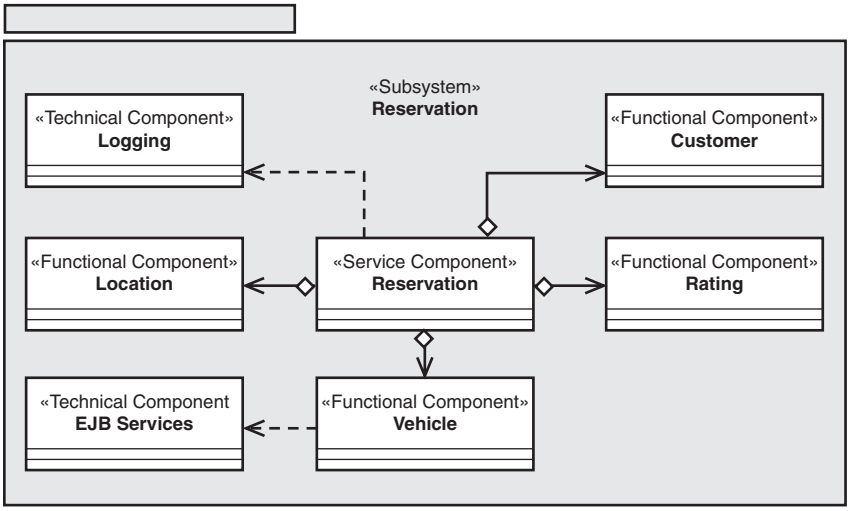



**Figure 4-6** The relationship between a subsystem and its constituent service, functional, and technical components

Identifying the various subsystems followed by the derivation of the service components and its constituent functional and technical components that together define a subsystem is the crux of this major step of subsystem analysis.

### 4.2.3.3 Component Specification

From here on, it is AAU work! There is nothing very special about SOA that we would do. Providing the detailed microlevel design is the focus of this step (that is, the software model in terms of class diagrams, sequence diagrams, and so on). Each service component identified and designed at a high level in the preceding step is further elaborated into a much more detailed treatment. Some typical microdesign steps that apply to a service component are the following:

1. Identify component characteristics, including component attribute and operations together with the policies and rules that they implement.
2. Identify events and messages that the components must sense and respond as a triggering mechanism. Incoming and outgoing component messages are also specified.
3. Identify internal component flow (representing the internal flow of control within the service component and which can be represented as a sequence or collaboration diagram).
4. Create component class diagrams, which are class models that show the static relationships between the functional and technical components.

Similar types of microdesign activities must be performed for each of the functional and technical components so that they are designed and specified to an adequate and unambiguous level of detail to be comfortably handed over to the implementation team for converting into working code. Because these are AAU activities, they are not covered in any further detail in this chapter.

Phew! That was a long section, but we did cover a lot of ground.

Okay, so what did we achieve in the service specification phase?

We were able to filter out only those services that are perfect candidates for exposure, and we did that by applying the service litmus test.

For each of the services tagged for exposure, we provided prescriptive guidance on how to design (at a macro level) a full specification for them, including the following:

- How services are dependent on each other (service dependencies)
- How services are orchestrated together to form composites that enable business processes (flows) (service compositions)
- Identifying and documenting the nonfunctional requirements that each service must implement and comply with (service NFRs)
- Detailed specification of the service messages (service message specification)
- Identifying and documenting the state requirements for each service (service state management)

While developing service message specifications, we acknowledged how these specifications tie in with and influence the information architecture, the integration architecture, and the data architecture of the system. Services are not the only SOA construct that we designed in this phase of SOMA. The processes were further elaborated and their flows were represented as an orchestration of services and IT components. The service component was also designed using their constituent functional and technical components.

From a service model standpoint, what have we achieved?

- Provided a service exposure rationale
- Addressed service dependencies
- Addressed service compositions and how they help in realizing process flows

- Emphasized the need to document service NFRs
- Addressed the recommended approach to define service messages
- Explained why it's necessary to document state management

Having achieved this, we move on to the third and last phase in this method: realization decisions for services.

#### 4.2.3.4 Realization for Services

The method in its first two phases not only demonstrated how to use a three-pronged approach for service identification but they also offered guidance about how to provide detailed specification for the services, service components, and process flows. The main focus of the method in this phase is to provide guidance about how to take architectural decisions that facilitate service realization. It is important to note that SOMA does not, at this point in time, address the actual implementation of services; instead, it provides enough information and detail so that an SOA implementation phase can just concentrate on the development and implementation of the services. Implementation is the phase wherein a specific technology, programming language and platform is chosen and used to transform the design specifications and the realization recipes and patterns into executable code.

This phase has three major activities: component allocation to layers, technical feasibility analysis, and realization decisions. The rest of this section focuses on these three major activities.

#### 4.2.3.5 Component Allocation to Layers

So far, this method has identified services, process flows, service components, functional components, and technical components. It also argued that technical components belong to a genre of components that do not directly provide business functionality but instead focus on delivering infrastructure functionalities that might be used by multiple functional and technical components. Keeping the solution stack in mind, we want to provide architectural recipes to allocate the SOA artifacts that we have identified, to the pertinent layers in the solution stack.

The service components and the functional components are all allocated to Layer 2 in the stack. The services, both atomic and composite, are allocated to Layer 3 in the stack. The process flows orchestrated using services from Layer 3 and functional and technical components from Layer 2, are allocated to Layer 4 of the stack. Technical components are of different types. There can be technical components that encapsulate a persistence framework or a data access layer. This type of technical component is usually allocated to Layer 8 (the data architecture layer). Some technical components encapsulate event management functionality, whereas others might provide queue management functionality, and some may encapsulate transaction management features. These types of components are allocated to Layer 6 (the integration layer). There can be other types of technical components, such as cache management, permissions management, audit management, and so forth. These types of components, which usually assist in complying with QoS requirements, are usually allocated to Layer 7 (the QoS layer). As you can see, based on the characteristics of each layer in the reference architecture, the method assists us to map the various types of software artifacts to the layers.

Without paying specific attention to the names of the components, specifically between Layers 1 and 4, Figure 4-7 depicts how different types of software building blocks, which the method assists us in identifying, are allocated to each layer in the solution stack.

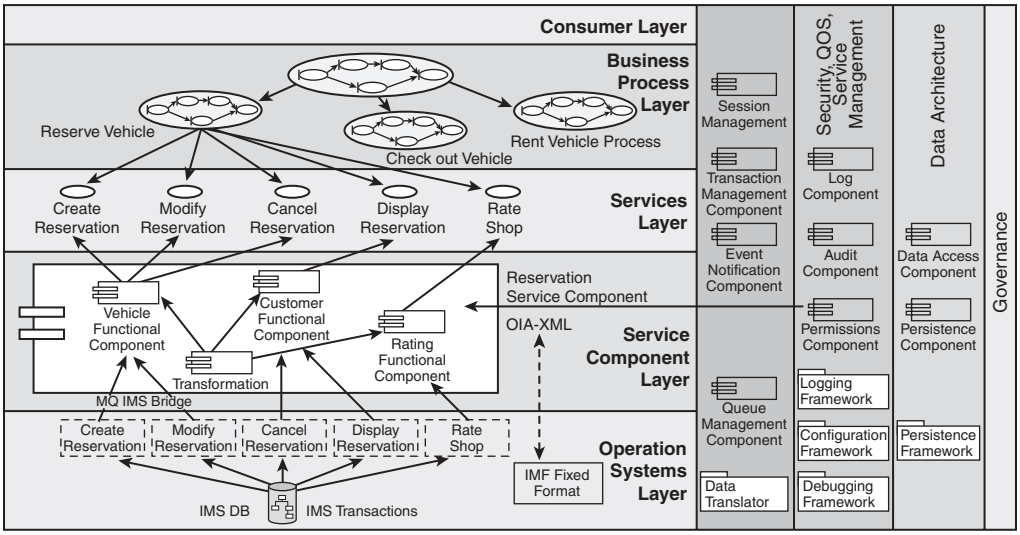



**Figure 4-7** Allocation of software artifacts on the layers of the SOA reference architecture

### 4.2.3.6 Component Allocation to Layers—Technical Feasibility Analysis

Technical feasibility takes input primarily from existing asset analysis and takes into account the services portfolio defined during service specification. The main focus of this activity is to perform a detailed analysis of existing systems to evaluate how much of the existing technology-specific implementation can be leveraged to realize the services. This activity, as is the SOMA method itself, is primarily iterative in nature, and it can be started as early as during the EAA technique during service identification. The functionality, along with the transactions identified during the early stages of service identification, needs to be validated for feasibility for componentization. This type of feasibility analysis results in architectural decisions for either the functional or operational architecture of the system. This is the step in the method where the deep-dive technical analysis of leveraging existing system functionality is actually formalized. You can do as much service specification as you want, but if you do not provide deep insight into the implementation of the service, ably supported by its justification, there still remains that proverbial gap between where architects hand off the design and where developers start with the exact knowledge on what to implement!

Let's consider an example of assessing the feasibility of using legacy system functionality. We must consider many technology aspects of legacy systems when looking to reuse such an existing asset for service realization. Some notable examples include the following:

- Exception handling in legacy systems is typically in the form of program termination. This might not be acceptable in a real-time SOA-based system.
- Authentication and authorization are often built in to the legacy application code using proprietary technologies and protocols. If legacy functionalities protected via security credentials are considered for reuse, there needs to be a mechanism to integrate the embedded security credentials in a federated environment. Externalizing the legacy security credentials might raise technological issues that will have to be addressed.
- The typical nightly batch processes for data synchronization or request submission may just be too infrequent to be used in real-time scenarios. In such cases, the legacy processing system might need to be amended; in extreme cases, the process might prove unusable.

These examples provide a snippet of the challenges that must be addressed when considering the reuse of existing systems and their functionality. Technical feasibility analysis addresses these types of issues by taking architectural decisions and justifying them for consideration.

#### 4.2.3.7 Realization Decisions

The technical feasibility analysis has a significant influence on how services ought to be realized in the best possible manner. Although technical feasibility analysis is initiated very early in the identification phase of the method and is performed throughout the various phases, considering all the various design and implementation alternatives, this step formalizes the final realization decision for each service and provides justification for the choice. This justification is a key step in the process and helps in maintenance and enhancement of the system in the years to come. The same realization alternatives could well be reanalyzed during enhancement of the system a few years down the road; and while doing so, the then-available technology might justify a different alternative as better suited. Thus, the snapshot in time of the architectural justification often proves invaluable.

In general, this method recommends considering six different high-level realization alternatives, as follows:

1. **Integrate**—Wrap existing legacy applications with SOA technologies and provide a service façade using open standards and protocols, for seamless integration into an SOA. Adapter technology is typically suited for this purpose.
2. **Transform**—Transform parts of legacy applications and expose them. This might involve the extraction of business rules or policies and rewriting the code in a modern SOA-aware programming language. This often falls under the discipline of legacy modernization.
3. **Buy**—Purchase products from independent service vendors (ISV) who provide out-of-the-box functionality that are exposed as services. Note that this option often results in a classic SOA antipattern in which the features of the ISV product often dictate the requirements of an enterprise. So do not fall into this trap and only evaluate the ISV functionality in the context of the project requirements.

4. **Build**—Build applications following the principles and best practices of SOA. This is often called the domain of custom application development.
5. **Subscribe**—Subscribe to external vendors who provide services that meet or exceed specific business functional requirements. Various SOA vendors are trying to find a niche in the market where they can specialize in a specific type of offering. Credit card authorization service is a classic example. Rarely would we see any enterprise developing this functionality indigenously. Instead, they subscribe to the best service provider that suits their needs.
6. **Outsource**—Outsource an entire part of the organization's functions to a third party. This is not yet considered mainstream because SOA is still undergoing some critical phases in its maturity and adoption. However, there are companies that specialize in, say, HR operations, and we have started seeing big corporations outsourcing an entire department. These third parties will provide services that need to be seamlessly integrated with the enterprise business processes.

This is what realization decisions help us achieve: a justification of the implementation mechanism for the services in the services portfolio.

So, what did we achieve in the realization phase?

- Understood how to allocate the various software building blocks onto the layers of the solution stack.
- Appreciated the justification to perform a detailed technical feasibility analysis before using existing legacy functionality for service realization.
- Identified the various options available for service realization.

And from a service model standpoint, what have we addressed?

- We were able to provide realization decisions for services.

This marks the completion of the three phases of the SOMA method. By now, I hope you can appreciate why a service-oriented analysis and design method, like SOMA, is required in any SOA-based initiative. The treatment provided here has hopefully demonstrated how SOMA is built on top of OOAD, while adding modeling and design techniques specific to SOA.

### 4.2.4 Using SOMA

The language of SOMA is crisp, clear, and very much focused on, providing a methodology to solve the challenges the IT community faces regarding service-oriented design. However, it is important to keep in mind that tools are integral to processes or methods, and a well-articulated method drives the development of tools in support of them. Rational Unified Process (RUP) has extended its process technique to incorporate service-oriented design, and it has used the SOMA method as the basis for its extension. This method extension is available as a plug-in in a product from Rational Software called the Rational Method Composer (see the "References" section).

SOMA method artifacts can also be expressed as a platform-independent model. Think of SOMA as providing a meta-language that helps in defining a service model. This model representation, defining not only the SOA constructs but also the relationships and constraints between them, is called the meta-model for SOMA. Any design tool that can implement the SOMA meta-model will be able to provide a tooling environment for service-oriented modeling and design based on the SOMA method. Although we do not provide the entire meta-model here, we do offer hints about how to develop one.

Hint: A *business domain* can be decomposed into one or more *functional areas*. A *functional area* can be decomposed into one or more *subsystems*. A *subsystem* contains one or more *service components*. A *service component* uses one or more *functional components* and *technical components*. If you try to enlist all the various constructs of SOMA and then model relationships between them, together with constraints on the relationships, you will be able to create the SOMA meta-model. It is then just a matter of implementing the meta-model in a software modeling tool! IBM has already developed a tool for SOMA and has been using it productively and successfully in multiple client engagements.

## 4.3 Conclusion

This chapter provided a detailed overview of a service-oriented design methodology, and we used the IBM-developed SOMA methodology as guidance on how to effectively develop an SOA-based system design.

The chapter also covered the SOA reference architecture, also called an SOA solution stack. It described each layer and identified the kind of software building blocks that constitute each layer. It then focused on how the SOMA method assists in the development of the first-class constructs of SOA: service, service components, and flows through the three phases of identification, specification, and realization.

As you finish this chapter, we hope that you now appreciate the need for a service-oriented design methodology and have learned how to execute the same via the SOMA methodology.

## 4.4 Links to developerWorks Articles

**A.4.1** Arsanjani, A. et al. Design an SOA Solution Using a Reference Architecture, IBM developerWorks, March 2007. www-128.ibm.com/developerworks/library/ar-archtemp/.

**A.4.2** Arsanjani, A. *Service Oriented Modeling and Architecture*, IBM developerWorks, November 2004. www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/. www-128.ibm.com/developerworks/architecture/library/ar-archtemp/.

## 4.5 References

Gamma, E. et al. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

Grossman, B. and J. Naumann. ACORD & XBRL US-XML Standards and the Insurance Value Chain. Acord.org, May 2004. www.arord.org/news/pdf/ACORD_XBRL.pdf.

Deblaere M. et al. IBM Insurance Application Architecture (IAA), White Paper, April 2002. www.baoxian119.com/xiazai/updownload/iaa2002whitepaper.pdf.

The complete and latest release of Justice XML Data Dictionary (JXDD). http://it.ojp.gov/jxdd/prerelease/3.0.0.3/index.html.

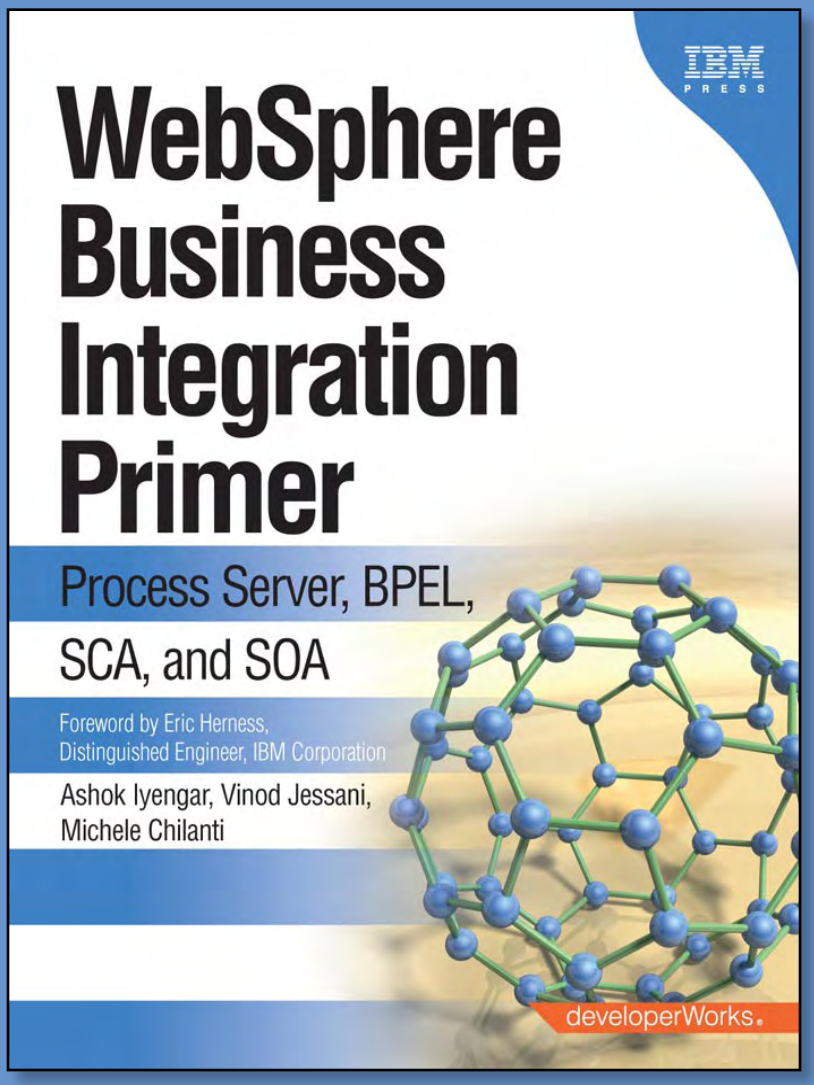
The official BPEL specifications are maintained by OASIS. www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

The official site that maintains the enhanced Telecom Operations Map (eTOM). www.tmforum.org/browse.aspx?catID=1647

Download the latest Open Travel Alliance (OTA) specifications from the opentravel site. www.opentravel.org/.

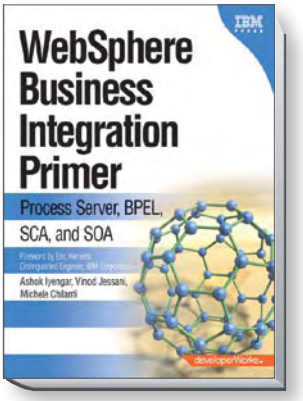
Download a trial version of Rational Method Composer. http://www14.software.ibm.com/webapp/download/product.jsp?id=TMMS-6GAMST&s=z&cat=&S_TACT=%26amp%3BS_CMP%3D&S_CMP=

Download the RUP plug-in for SOA. http://www.ibm.com/developerworks/rational/library/05/510_soaplug/



Ashok Iyengar
Vinod Jessani
Michele Chilanti






# WebSphere Business Integration Primer

## Process Server, BPEL, SCA, and SOA

©2008 ISBN: 0-13-224831-X

## Table of Contents



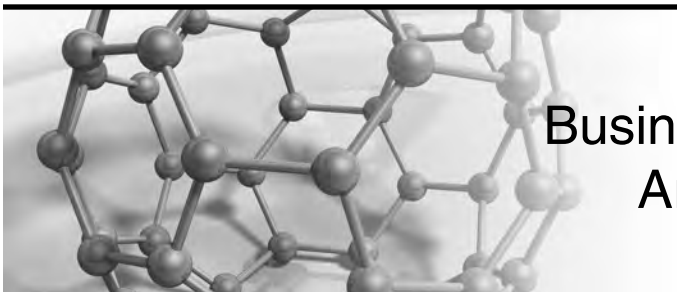
## About the Authors

**ASHOK IYENGAR** is a member of the WebSphere Enablement Team based in San Diego. He has worked extensively with the WebSphere software platform, and for the past four years, has concentrated on the WebSphere Business Integration suite. He also coauthored IBM WebSphere Portal Primer.

**VINOD JESSANI** is a senior software engineer with the IBM WebSphere Enablement Team in San Diego. He has more than 15 years' experience in distributed systems and online transaction processing systems.

**MICHELE CHILANTI** is a consulting IT specialist with IBM Software Services, where he consults daily with IBM customers worldwide, supporting their J2EE and WebSphere Business Integration development and deployment projects.

IBM Press™



# Chapter 2

# Business Integration Architecture and Patterns

A typical business integration project involves coordinating several different IT assets, potentially running on different platforms, and having been developed at different times using different technologies. Being able to easily manipulate and exchange information with a diverse set of components is a major technical challenge. It is best addressed by the programming model used to develop business integration solutions.

This chapter explores the fundamentals of the business integration programming model. It introduces the Service Component Architecture (SCA) and discusses patterns related to business integration. Patterns seem to permeate our lives. Sewing patterns, think-and-learn patterns for children, home construction patterns, wood-carving patterns, flight patterns, wind patterns, practice patterns in medicine, customer buying patterns, workflow patterns, design patterns in computer science, and many more exist.

Patterns have proven successful in helping solution designers and developers. Therefore, it is not surprising that we now have business integration patterns or enterprise integration patterns. In the referenced literature, you will find a wide array of patterns that are applicable to business integration, including patterns for request and response routing, channel patterns (such as publish/subscribe), and many more. Abstract patterns provide a template for resolving a certain category of problems, whereas concrete patterns provide more specific indications of how to implement a specific solution. This chapter focuses on patterns that deal with data and service invocation, which are at the foundation of the programming model of the IBM software strategy for WebSphere business integration.


A.2.1

## Business Integration Scenarios

Enterprises have many different software systems that they use to run their business. In addition, they have their own ways of integrating these business components. The two most prevalent business integration scenarios are as follows:

- **Integration broker:** In this use case, the business integration solution acts as an intermediary located among a variety of "back-end" applications. For example, you might need to ensure that when a customer places an order using the online order management application, the transaction updates relevant information in your Customer Relationship Management (CRM) back end. In this scenario, the integration solution needs to be able to capture and possibly transform the necessary information from the order management application and invoke the appropriate services in the CRM application.
- **Process automation:** In this scenario, the integration solution acts as the glue among different IT services that would otherwise be unrelated. For example, when a company hires an employee, the following sequence of actions needs to occur:
  - The employee's information is added to the payroll system.
  - The employee needs to be granted physical access to the facilities, and a badge needs to be provided.
  - The company might need to assign a set of physical assets to the employee (office space, a computer, and so on).
  - The IT department needs to create a user profile for the employee and grant access to a series of applications.

  Automating this process is also a common use case in a business integration scenario. In this case, the solution implements an automated flow that is triggered by the employee's addition to the payroll system. Subsequently, the flow triggers the other steps by creating work items for the people who are responsible for taking action or by calling the appropriate services.

In both scenarios, the integration solution needs to do the following:

- Work with disparate sources of information and different data formats, and be able to convert information between different formats
- Be able to invoke a variety of services, potentially using different invocation mechanisms and protocols

Throughout this book, we illustrate how the foundational programming model of IBM's WebSphere Process Server (WPS) addresses these requirements.

## Business Integration: Roles, Products, and Technical Challenges

Successful business integration projects require a few basic ingredients:

- A clear separation of roles in the development organization to promote specialization, which typically improves the quality of the individual components that are developed
- A common business object (BO) model that enables business information to be represented in a canonical format
- A programming model that strongly separates interfaces from implementations and that supports a generic service invocation mechanism that is totally independent of the implementation and that only involves dealing with interfaces
- An integrated set of tools and products that supports development roles and preserves their separation

The following sections elaborate on each of these ingredients.

## Clear Separation of Roles

A business integration project requires people in four collaborative, but distinctly separate, roles:

- **Business analyst:** Business analysts are domain experts responsible for capturing the business aspects of a process and for creating a process model that adequately represents the process itself. Their focus is to optimize the financial performance of a process. Business analysts are not concerned with the technical aspects of implementing processes.
- **Component developer:** Component developers are responsible for implementing individual services and components. Their focus is the specific technology used for the implementation. This role requires a strong programming background.
- **Integration specialist:** This relatively new role describes the person who is responsible for assembling a set of existing components into a larger business integration solution. Integration developers do not need to know the technical details of each of the components and services they reuse and wire together. Ideally, integration developers are concerned only with understanding the interfaces of the services that they are assembling. Integration developers should rely on integration tools for the assembly process.
- **Solution deployer:** Solution deployers and administrators are concerned with making business integration solutions available to end users. Ideally, a solution deployer is primarily concerned with binding a solution to the physical resources ready for it to function (databases, queue managers, and so on) and not with having a deep understanding of the internals of a solution. The solution deployer's focus is quality of service (QoS).

## A Common Business Object Model

As we discussed, the key aspects of a business integration project include the ability to coordinate the invocation of several components and the ability to handle the data exchange among those. In particular, different components can use different techniques to represent business items such as the data in an order, a customer's information, and so on. For example, you might have to integrate a Java application that uses entity Enterprise Java Beans

(EJBs) to represent business items and a legacy application that organizes information in COBOL copybooks. Therefore, a platform that aims to simplify the creation of integration solutions should also provide a generic way to represent business items, irrespective of the techniques used by the back-end systems for data handling. This goal is achieved in WPS and WebSphere Enterprise Service Bus (WESB) thanks to the *business object framework*.

The business object framework enables developers to use XML Schemas to define the structure of business data and access and manipulate instances of these data structures (business objects) via XPath or Java code. The business object framework is based on the Service Data Object (SDO) standard.

### The Service Component Architecture (SCA) Programming Model

The SCA programming model represents the foundation for any solution to be developed on WPS and WESB.

SCA provides a way for developers to encapsulate service implementations in reusable components. It enables you to define interfaces, implementations, and references in a technology-agnostic way, giving you the opportunity to bind the elements to whichever technology you choose.

There is also an SCA client programming model that enables the invocation of those components. In particular, it enables runtime infrastructures based on Java—such as IBM's WebSphere Process Server, BEA's WebLogic Server (with its Aqualogic product family), and Oracle's Application Server (part of Oracle's Fusion Middleware family)—to interact with non-Java runtimes. SCA uses business objects as the data items for service invocation.

### Tools and Products

IBM's WebSphere Integration Developer is the integrated development environment that has all the necessary tools to create and compose business integration solutions based on the technologies just mentioned. These solutions typically are deployed to the WPS or, in some cases, to the WESB—the products that are at the center of this book.

Now that you understand the key ingredients of business integration solutions, let's take a look at the business object framework, at SCA, and at some of the key patterns, processes, and qualifiers in more detail.

## The Business Object Framework

The computer software industry has developed several programming models and frameworks that enable developers to encapsulate business object information. In general, a BO framework should provide database independence, transparently map custom business objects to database tables, and bind business objects to user interfaces. Of late, XML schemas are perhaps the most popular and accepted way to represent the structure of a business object.

From a tooling perspective, WebSphere Integration Developer (WID) provides developers with a common BO model for representing different kinds of entities from different domains. At development time, WID represents business objects as XML schemas. At runtime, however, those same business objects are represented in memory by a Java instance of an SDO. SDO is a standard specification that IBM and BEA Systems have jointly developed and agreed on. IBM has extended the SDO specification by including some additional services that facilitate the manipulation of data within the business objects. We'll discuss some of these later in this chapter.

Before we get into the BO framework, let's look at the basic types of data that get manipulated:

- **Instance data** is the actual data and data structures, from simple, basic objects with scalar properties to large, complex hierarchies of objects. This also includes data definitions such as a description of the basic attribute types, complex type information, cardinality, and default values.
- **Instance metadata** is instance-specific data. Incremental information is added to the base data, such as change tracking (also known as change summary), context information associated with how the object or data was created, and message headers and footers.
- **Type metadata** is usually application-specific information, such as attribute-level mappings to destination enterprise information system (EIS) data columns (for example, mapping a BO field name to a SAP table column name).
- **Services** are basically helper services that get data, set data, change summary, or provide data definition type access.

Table 2.1 shows how the basic types of data are implemented in the WebSphere platform.

**Table 2.1** Data Abstractions and the Corresponding Implementations

| Data Abstraction | Implementation |
|---|---|
| Instance data | Business object (SDO) |
| Instance metadata | Business graph |
| Type metadata | Enterprise metadata |
| | Business object type metadata |
| Services | Business object services |

### Working with the IBM Business Object Framework

As we mentioned, the WPS BO framework is an extension of the SDO standard. Therefore, business objects exchanged between WPS components are instances of the *commonj.sdo.DataObject* class. However, the WPS BO framework adds several services and functions that simplify and enrich the basic *DataObject* functionality.

To facilitate the creation and manipulation of business objects, the WebSphere BO framework extends SDO specifications by providing a set of Java services. These services are part of the package named *com.ibm.websphere.bo*:

- **BOFactory:** The key service that provides various ways to create instances of business objects.
- **BOXMLSerializer:** Provides ways to "inflate" a business object from a stream or to write the content of a business object, in XML format, to a stream.
- **BOCopy:** Provides methods that make copies of business objects ("deep" and "shallow" semantics).
- **BODataObject:** Gives you access to the data object aspects of a business object, such as the change summary, the business graph, and the event summary.
- **BOXMLDocument:** The front end to the service that lets you manipulate the business object as an XML document.
- **BOChangeSummary and BOEventSummary:** Simplifies access to and manipulation of the change summary and event summary portion of a business object.
- **BOEquality:** A service that enables you to determine whether two business objects contain the same information. It supports both deep and shallow equality.
- **BOType and BOTypeMetaData:** These services materialize instances of *commonj.sdo.Type* and let you manipulate the associated metadata. Instances of *Type* can then be used to create business objects "by type."

Chapter 4, "WebSphere Integration Developer," introduces the facilities that WID provides to enable you to quickly formulate your object definitions.

## Service Component Architecture


A.2.2

SCA is an abstraction you can implement in many different ways. It does not mandate any particular technology, programming language, invocation protocol, or transport mechanism. SCA components are described using Service Component Definition Language (SCDL), which is an XML-based language. You could, in theory, create an SCDL file manually. In practice, you're more likely to use an integrated development environment (IDE) such as WebSphere Integration Developer to generate the SCDL file.

An SCA component has the following characteristics:

- It wraps an implementation artifact, which contains the logic that the component can execute.
- It exposes one or more interfaces.

- It can expose one or more references to other components. The implementation's logic determines whether a component exposes a reference. If the implementation requires invoking other services, the SCA component needs to expose a reference.

This chapter focuses on the SCA implementation that WPS offers and the WID tool that is available to create and combine SCA components. WPS and WID support the following implementation artifacts:

- Plain Java objects
- Business Process Execution Language (BPEL) processes
- Business state machines
- Human tasks
- Business rules
- Selectors
- Mediations

SCA separates business logic from infrastructure so that application programmers can focus on solving business problems. IBM's WPS is based on that same premise. Figure 2.1 shows the architectural model of WPS.

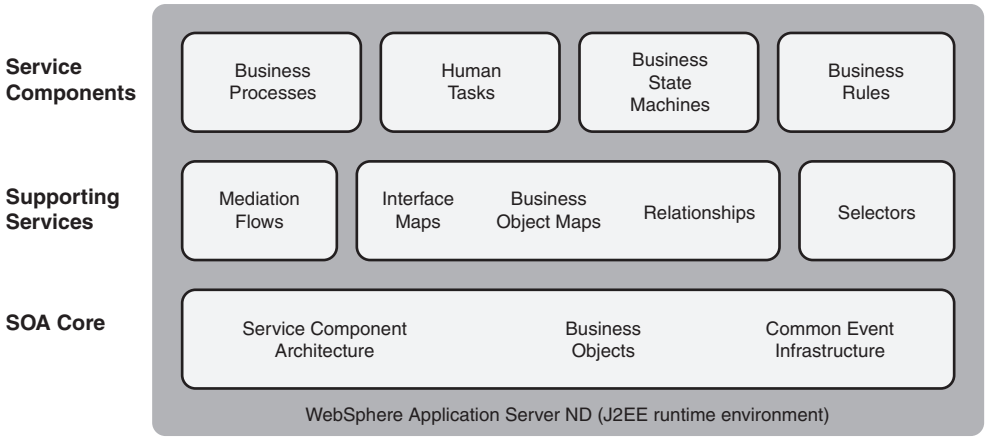



**Figure 2.1** Architectural model for WPS

In the WebSphere environment, the SCA framework is based on the Java 2 Platform, Enterprise Edition (J2EE) runtime environment of WebSphere Application Server. The overall WebSphere Process Server framework consists of SOA Core, Supporting Services, and the Service Components. The same framework with a subset of this overall capability, targeted more specifically at the connectivity and application integration needs of business integration, is available in WESB.

The interface of an SCA component, as illustrated in Figure 2.2, can be represented as one of the following:

- A Java interface
- A WSDL port type (in WSDL 2.0, port type is called interface)

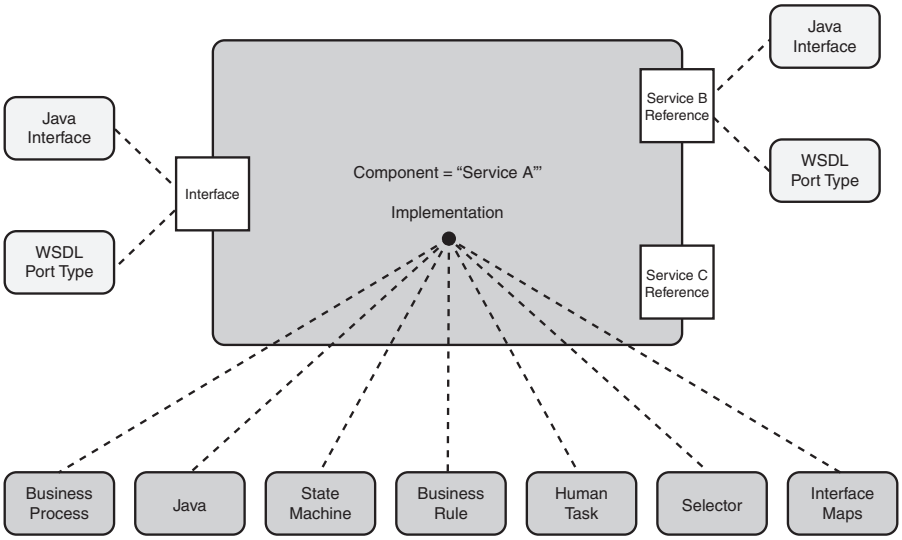



**Figure 2.2** SCA in WPS

An SCA module is a group of components wired together by directly linking references and implementations. In WID, each SCA module has an *assembly diagram* associated with it, which represents the integrated business application, consisting of SCA components and the wires that connect them. One of the main responsibilities of the integration developer is to create the assembly diagram by connecting the components that form the solution. WID provides a graphical Assembly Editor to assist with this task. When creating the assembly diagram, the integration developer can proceed in one of two ways:

- **Top-down** defines the components, their interfaces, and their interactions before creating the implementation. The integration developer can define the structure of the process, identify the necessary components and their implementation types, and then generate an implementation skeleton.
- **Bottom-up** combines existing components. In this case, the integration developer simply needs to drag and drop existing implementations onto the assembly diagram.

The bottom-up approach is more commonly used when customers have existing services that they want to reuse and combine. When you need to create new business objects from scratch, you are likely to adopt the top-down approach. Chapter 4 introduces the various

wizards in WID and lays out the six phases of creating a simple module using the top-down approach.

## The SCA Programming Model: Fundamentals

The concept of a software *component* forms the basis of the SCA programming model. As we mentioned, a component is a unit that implements some logic and makes it available to other components through an interface. A component may also require the services made available by other components. In that case, the component exposes a *reference* to these services.

In SCA, every component must expose at least one interface. The assembly diagram shown in Figure 2.3 has three components—C1, C2, and C3. Each component has an interface that is represented by the letter I in a circle. A component can also refer to other components. References are represented by the letter R in a square. References and interfaces are then linked in an assembly diagram. Essentially, the integration developer "resolves" the references by connecting them with the interfaces of the components that implement the required logic.

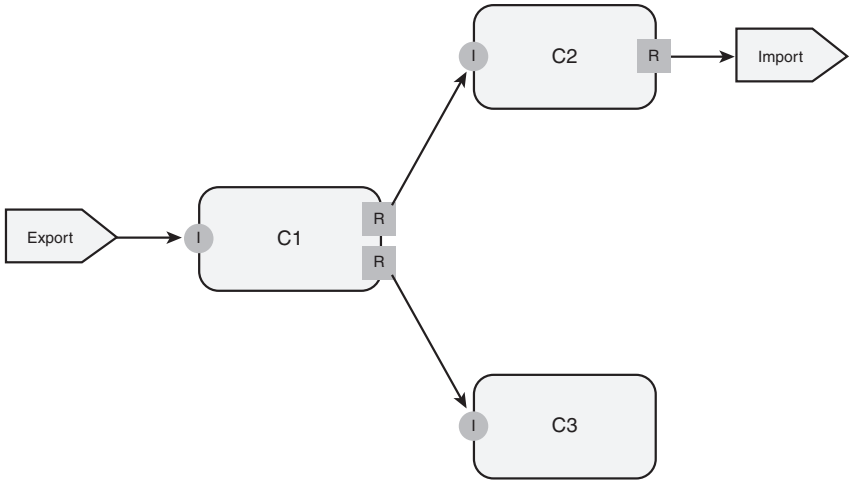



**Figure 2.3** Assembly diagram

## Invoking SCA Components

To provide access to the services to be invoked, the SCA programming model includes a *ServiceManager* class, which enables developers to look up available services by name. Here is a typical Java code fragment illustrating service lookup. The ServiceManager is used to obtain a reference to the BOFactory service, which is a system-provided service:

```
//Get service manager singleton
ServiceManager smgr = new ServiceManager();
//Access BOFactory service
BOFactory bof =(BOFactory)
        mgr.locateService("com/ibm/websphere/bo/BOFactory");
```

Developers can use a similar mechanism to obtain references to their own services by specifying the name of the service referenced in the *locateService* method. We illustrate in more detail the usage of the SCA programming model in Chapter 11, "Business Integration Programming." For the time being, we want to emphasize that after you have obtained a reference to a service using the *ServiceManager* class, you can invoke any of the available operations on that service in a way that is independent of the invocation protocol and the type of implementation.

SCA components can be called using three different invocation styles:

- **Synchronous invocation:** When using this invocation style, the caller waits synchronously for the response to be returned. This is the classic invocation mechanism.
- **Asynchronous invocation:** This mechanism allows the caller to invoke a service without waiting for the response to be produced right away. Instead of getting the response, the caller gets a "ticket," which can be used later to retrieve the response. The caller retrieves the response by calling a special operation that must be provided by the callee for this purpose.
- **Asynchronous invocation with callback:** This invocation style is similar to the preceding one, but it delegates the responsibility of returning the response to the callee. The caller needs to expose a special operation (the callback operation) that the callee can invoke when the response is ready.

## Imports

Sometimes, business logic is provided by components or functions that are available on external systems, such as legacy applications, or other external implementations. In those cases, the integration developer cannot resolve the reference by connecting a reference to a component containing the implementation he or she needs to connect the reference to a component that "points to" the external implementation. Such a component is called an *import*. When you define an import, you need to specify how the external service can be accessed in terms of location and the invocation protocol.

## Exports

Similarly, if your component has to be accessed by external applications, which is quite often the case, you must make it accessible. That is done by using a special component that exposes your logic to the "outside world." Such a component is called an *export*. These can also be invoked synchronously or asynchronously.

### Stand-alone References

In WPS, an SCA service module is equivalent to a J2EE EAR file and contains several other J2EE submodules. J2EE elements, such as a WAR file, can be packaged along with the SCA module. Non-SCA artifacts such as JSPs can also be packaged together with an SCA service module. This lets them invoke SCA services through the SCA client programming model using a special type of component called a stand-alone reference.

The SCA programming model is strongly declarative. Integration developers can configure aspects such as transactional behavior of invocations, propagation of security credentials, whether an invocation should be synchronous or asynchronous in a declarative way, directly in the assembly diagram. The SCA runtime, not the developers, is responsible for taking care of implementing the behavior specified in these modifiers. The declarative flexibility of SCA is one of the most powerful features of this programming model. Developers can concentrate on implementing business logic, rather than focusing on addressing technical aspects, such as being able to accommodate asynchronous invocation mechanisms. All these aspects are automatically taken care of by the SCA runtime. The declarative aspects of SCA programming are discussed in Chapter 11.

## Business Integration Patterns

Patterns help architects, designers, and developers use a common vocabulary to efficiently describe their solutions. This section discusses patterns at the business integration level, which fall into two classes: intraenterprise and interenterprise. These break down into Enterprise Application Integration (EAI) scenarios and business-to-business (B2B) scenarios. The EAI patterns deal mainly with application integration and database replication, whereas the B2B patterns deal with data exchange and process integration.

Another classification of business integration or Enterprise Integration Pattern is based on messaging architectures and messaging specifications such as Java Messaging Service (JMS) and Web services. The common categories in this case are channel patterns, endpoint patterns, routing patterns, system management patterns, and transformation patterns.

A.2.3

### Data Exchange Patterns

A data exchange pattern is a rather simple pattern that is predicated on an agreement to use a common data format. The idea behind the SDO standard is to enable Java applications to support the following three common data exchange patterns. Actually, these are mechanisms that you can implement using different design patterns:

- The Plain Business Object pattern
- The Disconnected Object pattern
- The Event pattern

### The Plain Business Object Pattern

The Plain Business Object pattern, also known as the Document pattern, is the most common data exchange mechanism. A client component populates a business object and then submits the object to a service by invoking a specific operation. Chapter 4 discusses defining and using plain business objects.

### The Disconnected Object Pattern

The Disconnected Object pattern or the Transfer Object pattern is another common data exchange mechanism. Its primary purpose is to minimize the amount of time an application needs to be connected to the back end when making changes to an object. It extends the Transfer Object pattern of the Model-View-Controller architecture.

The Disconnected Object pattern enables an application to get hold of data from a back-end system and manipulate the data without requiring a connection to the back end. Then the application makes the changes persistent in a separate transaction that involves connectivity with the back end. The pattern extends this concept by introducing a mechanism to track the changes made to the business object or to any of the business objects contained by that business object.

In the Disconnected Object pattern, a system uses the change history in addition to the information stored in the object itself. The mechanism to track changes is called a business graph (BG), and it essentially includes four things:

- A *copy of the business object data* that can be manipulated even when no connection exists to the repository that holds the business object itself
- A *change history*, which stores the values of the business object before any manipulation occurred (for example, the original values), with an indication of the operations that modified the state of the business object
- An *event summary*, which contains the identifiers of objects affected by a change and event information recording the actual data involved in the business transaction
- A *verb*, which specifies the kind of operation that was performed (for example, a create or delete operation)

### The Event Pattern

The Event pattern is the data exchange mechanism used when an event is produced by an EIS and is injected into WPS through an adapter. Typically, an adapter captures an event that occurred in a back-end system and creates the appropriate business object within WPS. For instance, in a travel reservation application that maintains profiles for travelers, a user might add a new airline company to the list of preferred airlines in his or her traveler's profile. Using the event pattern, that operation is captured as an "update" event that involves a certain traveler ID and a certain airline ID. The appropriate business objects will be materialized in WPS for the benefit of the business processes that can manipulate them.

The Event pattern makes use of the "delta image" information, which is a recording of what was changed in the original object. In our example, the addition of an airline to the list of

preferred airlines would be the delta image. The Event pattern becomes important when you design complex relationships between different back ends, especially in integration scenarios that require keeping information about equivalent entities, managed by different applications, synchronized.

**Object Mapping Pattern**

Every so often, you will hear the term GBO and ASBO. GBO stands for Generic Business Object, and ASBO stands for Application-Specific Business Object. This terminology did not catch on, but we see ASBO-to-GBO and GBO-to-ASBO patterns all the time in business integration solutions.

Many patterns have limitations, which are overcome by the use of a complementary or associated pattern. The Process Integration pattern takes the limitations raised by the Data Exchange pattern and addresses them by providing Business Process Integration (BPI) services. The common underlying entity in both cases is the exchange of XML-based documents, which permits richer, more complex relationships.

## Business Processes

Business processes—specifically, BPEL-based business processes—form the cornerstone of service components in the SCA. Whether it is a simple order approval or a complex manufacturing process, enterprises have always had business processes. A business process is a set of activities, related to the business, that are invoked in a specific sequence to achieve a business goal. In the business integration world, a business process is defined using some kind of markup language.

These business processes can invoke other supporting services or contain other service components such as business state machines, human tasks, business rules, or data maps. And, when deployed, these processes can either get done quickly or run over a long period of time. Sometimes, these processes can run for years.

Like most components in the J2EE world, business processes run in a container. In IBM's WebSphere platform, this specific container is called the Business Process Choreographer, which we talk about in Chapter 9, "Business Integration Clients." The container or the BPEL engine provides all the services and process lifecycle requirements.

## Qualifiers

In a word, qualifiers are rules. Qualifiers define how much management should be provided by WPS for a component at runtime. A process application communicates its QoS needs to the WPS runtime environment by specifying service qualifiers. The qualifiers govern the

interaction between a service client and a target service. Qualifiers can be specified on service component references, interfaces, and implementations and are always external to an implementation. The different categories of qualifiers include the following:

- Transaction, which specifies the way transactional contexts are handled in an SCA invocation
- Activity session, which specifies how Activity Session contexts are propagated. An Activity Session extends the concept of transaction, to encompass a number of related transactions.
- Security, which specifies the permissions
- Asynchronous reliability—rules for asynchronous message delivery

SCA allows these QoS qualifiers to be applied to components declaratively (without requiring programming or a change to the services implementation code). This is done in WID. Table 2.2 lists the different types of qualifiers, along with their qualifying values. Usually, you apply QoS qualifiers when you are ready to consider solution deployment. Look for more details about QoS, especially event sequencing, in Chapter 11.

**Table 2.2** Adding References to the Application Deployment Descriptor

| Name | Qualifier Values |
|---|---|
| Reference qualifiers | Asynchronous reliability<br>Suspend transaction<br>Asynchronous invocation<br>Suspend activity session |
| Interface qualifiers | Event sequencing<br>Join activity session<br>Join transaction<br>Security permission |
| Implementation qualifiers | Activity session<br>Transaction<br>Security identity |

## Closing the Link

Imports, exports, references, and so on are new terms in this whole new paradigm called the Service Component Architecture (SCA), which was introduced in this chapter. From the developer's perspective, services are packaged in a service module, which is the basic unit of deployment and administration in an SCA runtime.

Service imports are used in a service module to use external services that are not part of the module itself (for example, services exported by other modules, stateless session EJBs, Web services, EIS services, and so on). External services that are referenced by import declarations are valid targets of service wires. The import binding definition does not need to be finalized at development time. Aspects such as the actual endpoint of services to invoke can be late-bound at deployment, administration, or even runtime. Service exports, on the other hand, are used to offer services from the service module to the outside world, such as services for other service modules or as Web services.

The following chapters build on these concepts and use the tooling and runtime examples to fully explain SCA. It is imperative that you understand the SCA model, because it forms the basis of business integration discussed in this book. It is beneficial to reiterate that patterns are not inventions or edicts; they are harvested from repeated experiences from and use by practitioners. Business integration does not always involve business processes, but in a lot of cases, business processes form the centerpiece of integration.

## Links to developerWorks

**A.2.1** www.ibm.com/developerworks/podcast/websphere/ws-soa2progmod.html

**A.2.2** www.ibm.com/developerworks/websphere/library/techarticles/0610_redlin/0610_redlin.html

**A.2.3** www.ibm.com/developerworks/websphere/techjournal/0508_simmons/0508_simmons.html

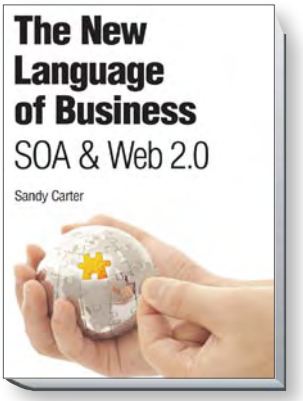
# The New Language of Business

## SOA & Web 2.0

Sandy Carter

Sandy Carter

BUY ME AT 35% OFF




# The New Language of Business

## SOA & Web 2.0

©2008 ISBN: 0-13-195654-X

## Table of Contents



## About the Author

**SANDY CARTER** is Vice President, SOA & WebSphere Strategy, Channels and Marketing for IBM Corporation. Sandy is responsible for driving IBM's cross-company, worldwide SOA marketing initiatives, and in this role, helps oversee the company's SOA strategy across software, services and hardware and sets the company's SOA marketing direction. has played a critical role in helping to identify SOA acquisition targets and ensure the successful integration of these organizations into the IBM SOA portfolio. Additionally, she directs SOA messaging and content, leading a global team in driving customer demand for IBM and IBM Business Partner SOA solutions. Sandy is a frequent speaker at industry events and has a leading blog in the industry for SOA (www-03.ibm.com/developerworks/blogs/page/SOA_Off_the_Record).

IBM Press™

CHAPTER

# 11

# Putting It All Together

I am on the North Carolina coast today, and I am surprised to learn that more than 2,000 shipwrecks have occurred off these coasts. Why so many? Hurricanes, treacherous shoals, unpredictable weather, and war caused the majority of the wrecks. My family and I rented jet skis and went exploring around the area to check out and learn about the history. It was neat to see how many other captains had "learned" about the ways to predict and avoid such hazards in the area and were able to successfully reach the North Carolina shore.

In the same way, this final chapter helps companies create a plan for their journey toward innovation. A set of guiding principles and goals is the focus of this chapter as you continue your flex-pon-sive* journey.

## GROWTH, BUSINESS FLEXIBILITY, AND INNOVATION ARE THE RESULTS OF A FLEX-PON-SIVE* COMPANY

In some ways, today's business environment is similar to the Internet era, when in the rush to embrace the Internet and to get a competitive edge, companies became preoccupied with e-commerce. In fact, instead of imagining a hybrid world, everyone said that "clicks would replace bricks" and that retail would be changed forever. Others thought that it was about more than transactions; they saw the larger vision of e-business. E-business was a new way of doing business. You can see the similarities to where we are today—the rush to adopt emerging technology and the misconceptions that business changes suddenly instead of gradually.

And today there is a bigger world emerging on the horizon. From the work that IBM has been doing with business leaders and from its client engagements, they have produced a study by the Institute for Business Value (IBV) on the business value of flexibility and SOA in this new world. The results showed that those companies moving to the flex-pon-sive* world were seeing the results in flexibility from SOA; 97% justified their first SOA project based on cost, 100% saw increased business flexibility, and 51% showed increased revenue growth. The 30 customers who produced these results are plunging into today's innovative and flex-pon-sive* world with an understanding that was lost in the e-business world. This new model requires that businesses change, but at an incremental pace. This study complements the CEO study that we analyzed at the start of this book, which showed that companies were primarily pursuing growth again and only secondly cost-cutting. Since that study was completed, competitive pressures have only increased—due to advances in technology, the rapid advent of globalization, and the consequent flat world. If there's any change, it's the insertion of an important qualifier—profitability. Profitable growth is now at the top of the list.

And in a follow-up study, our recently released Global CEO Study 2006, 765 CEOs in every major industry told us that the pressures

to achieve profitable growth had introduced a new mandate—the need to innovate:

- Two-thirds of the CEOs believe their organizations will need to introduce fundamental, radical changes in the next two years to respond to competitive pressures and external forces.
- Fewer than half say they've managed this magnitude of change successfully in the past.

With the growing sophistication about how and where innovation occurs, companies know that business flexibility is the driver. New ideas don't just come from inside their company, but from wikis, blogs, partners, customers, and even competitors. This world requires collaboration to solicit the ideas and flexibility to respond to those ideas. The insight is that CEOs now say that more of their ideas for innovation come from partners and clients than from their own employees.

The interesting commonality here is that all these new ideas come from some sort of collaboration, but to act on those ideas, business flexibility must be a number-one priority.

Among all the CEO areas of focus we examined, business flexibility and collaboration showed the clearest correlation with financial performance, whatever the financial metric—revenue growth, operating margin growth, or average profitability over time. Beyond product or service innovation, more CEOs are looking to business process innovation as a key competitive advantage. As one CEO put it, "Products and services can be copied. The business process and the model is the differentiator." Another CEO commented that new product introductions in his industry offered only one month of market exclusivity before they are duplicated in the marketplace.

This whole discussion is key because it shows that a few of the top areas we need to tackle are the alignment of business and IT, especially around joint goals, and a focus on those processes that will allow companies to differentiate themselves.

This book helps address these key questions:

- What are your company's business goals, and how do you align your whole company, including business and IT, around those goals?
- What governance mechanisms and mandates do you have in place to drive those goals throughout your corporation? Chapter 6, "SOA Governance and Service Lifecycle," addressed how to think through governance, one of the most important indicators for success.
- What flexibility and innovation are needed for those goals to be reached?
- What business processes need innovation to be successful? Chapter 3, "Deconstructing Your Business: Component Business Model," detailed one method to determine the core processes that you should focus on for success.
- How does your company create an environment of innovation and the power to act upon it?

A company cannot continue to succeed if it comes up with some superb ideas through powerful focus and collaboration, but fails to act upon them or is not flexible enough to respond quickly to market forces. Governance and a focus on the right processes coupled with flexibility to act are all critical for a flex-pon-sive* company.

So the bottom line is that companies must have change to innovate. Given that every business is so tied to technology, this conclusion places a premium on the underlying technology that runs your company.

## NOW, HOW DO YOU CONVINCE THE BUSINESS?

Behind every successful service oriented architecture (SOA) is the Business. With its promise of using existing technology to more closely align information technology (IT) with business goals, we have seen that SOAs have proven to help companies realize greater efficiencies, cost savings, and productivity.

Still, as many IT managers have learned, without executive endorsement, an SOA will be relegated to the confines of IT as opposed to being recognized as an organization-wide business strategy. While no two organizations are exactly alike, there are consistent themes that arise when aiming for approval to build an SOA.

For those many IT leaders who are facing the seemingly daunting challenge of presenting the importance and value of an SOA strategy to the executive suite, following are ten tips for selling SOA to the Business Leader. A few tips.

1. **Don't call it SOA**: Explain the value and benefits in business terms that reflect the organization's goals such as cost reduction, productivity, competitive advantage, etc. before diving into a technical conversation.
2. **Vision, not version**: Outline the immediate and long-term results from this strategy while avoiding discussions about specific version numbers.
3. **Build consensus throughout the company**: Prove the value of SOA through small, test projects conducted with volunteer departments in the organization. Make sure to include those department leaders when you later roll out the SOA.
4. **Start small yet live large**: When selecting those small test projects, choose to integrate and automate those business processes that can have the most widespread, positive impact across the organization.
5. **Ixnay on the TLA**: While it's easy to get caught up in the technical jargon that is fully understood among peers, remember that three letter acronyms (TLA) can sound as eloquent as pig Latin when trying to convince your CEO of a major, new strategic undertaking.
6. **Get to the powerful points**: Without relying on complex slides that can deter from the true purpose of the meeting.
7. **Conviction and prediction**: Articulate goals for each step along the SOA path. By publicly stating and achieving realistic goals

for the organization based on an SOA—increasing productivity or decreasing costs by XX percent—you can bolster confidence in the project and overall strategy.

8. **Reference third party validation (see the next section in detail!)**: Cite analyst data on the growth and adoption of service oriented architectures and point to relevant SOA success stories within your industry (and by your competitors).
9. **The close**: SOA what? Outline specific before-and-after scenarios of the impact of SOA on your particular organization to help disarm any naysayer and gain CEO approval.
10. **Qualify and quantify**: Set goals, track performance, and refine methodologies at every step along the way. Be sure to share the results with interested parties on a regular basis to demonstrate the success of your company's SOA journey.

The opportunity to evangelize SOA to company executives is rare. To make the most of your extended elevator pitch, remember to articulate business benefits, reiterate bottom line results, and illustrate the company-wide value of an SOA.

## SOA AND WEB 2.0 BECOME THE ENABLERS

A flexible business—a flex-pon-sive* business—requires flexible IT. Innovation requires change and SOA makes it easier for companies to change. Given this focus on business flexibility, growth, and innovation, the technology that most expedites these business goals is service oriented architecture (SOA). According to most of the analyst firms, SOA will become the de facto standard for business flexibility and collaboration among companies.

As we discussed in this book, SOA is all about an approach that views a business as linked services and considers the outcomes they bring. Because it is built on open standards, it is a way for businesses to tap into their existing technology investments and flexibly link previously fragmented data and business processes, creating a more complete view of operations, potential bottlenecks, and areas for growth.

As we learned, advances in open standards and software-development tools have made SOA applications easier to develop.

However, this does not mean that everyone is deploying SOA applications; the market is at the early stages of adoption. Services that join together to support business processes within SOA are designed in such a way that different parts can operate independently of one another. Because of this, any one feature can be changed without breaking other parts of the application. This makes companies that have adopted principles of SOA much more responsive to changing business requirements than those that rely on traditional software development, with one feature change potentially derailing an entire application.

The companies that master SOA technology operate more efficiently than their competitors and adapt more quickly to changing business conditions in their industries. And as we discussed earlier, Web 2.0 facilitates the collaboration aspects, and SOA enables the infrastructure for flexibility.

A great example is a retailer deciding whether to issue a credit card to a customer. It could use the technology to tap different sources and pull together information on a customer's creditworthiness and buying habits. A bank can use the same computing services to handle account transfer requests, whether they are coming from a teller, an ATM, or a Web application, avoiding the need for multiple applications. A manufacturer could measure more closely what is happening in its production process and then make adjustments that feed back instantly through its chain of suppliers.

SOA enables profitability through revenue growth and cost cutting. SOA enables innovation through collaboration and flexibility.

Your checklist for becoming a flex-pon-sive* business should include the following:

- Understand SOA and Web 2.0. Chapters 3 and 4, "SOA as the DNA of a Flex-pon-sive* and Innovative Company," start to articulate what you need to consider, but the goal of this book is not to make you technology experts. Rather, the goal is to provide you with enough information to ask the right questions to begin your journey.

- Develop the skills needed to embrace these new technologies.
- Understand the business implications of the new technologies.

## LEARNING FROM OTHER COMPANIES IS CRITICAL AROUND THE ENTRY POINTS

The companies that master SOA technology can operate more efficiently than their competitors and can more quickly adapt to changing business conditions in their industries. Meeting innovation priorities requires the ability to change flexibly, and companies should take a business-centric view of SOA (as opposed to an IT-centric view) to achieve these innovation goals (see Figure 11.1). As discussed in Chapter 4, "SOA as the DNA of a Flex-ponsive* and Innovative Company," a recent study of more than 500 companies conducted by Mercer Management Consultants showed that these companies are approaching SOA from entry points of people, process, and information, or all three. The lessons learned from the SOA entry points are furthered by the IBV study about SOA business value. This study of approximately 30 customers reveals some other lessons about revenue growth and cost cutting. 51% of the clients interviewed for this study expected their SOA deployment to grow their revenue, primarily by unlocking the potential of an existing process. To explore this in a real-world setting, review a bank's processes, such as a residential mortgages system, credit card system, or loan-servicing system. Following the IBM case study, an evaluation of those processes should reveal reusable parts, such as "submit loan application," "perform credit check," "determine credit line," or "calculate interest rate." SOA enables IT to recombine these reusable parts to create new products, such as a tailored home equity line of credit. With SOA, the business strategist is free to innovate.

Companies that started from one of these entry points have stories to illustrate the lessons that can be learned from other companies' experiences. Enterprise transformation powered by an SOA is really the holy grail the customer seeks. This enterprise transformation can begin with a set of entry point projects as a way for customers to start their transformation journey.

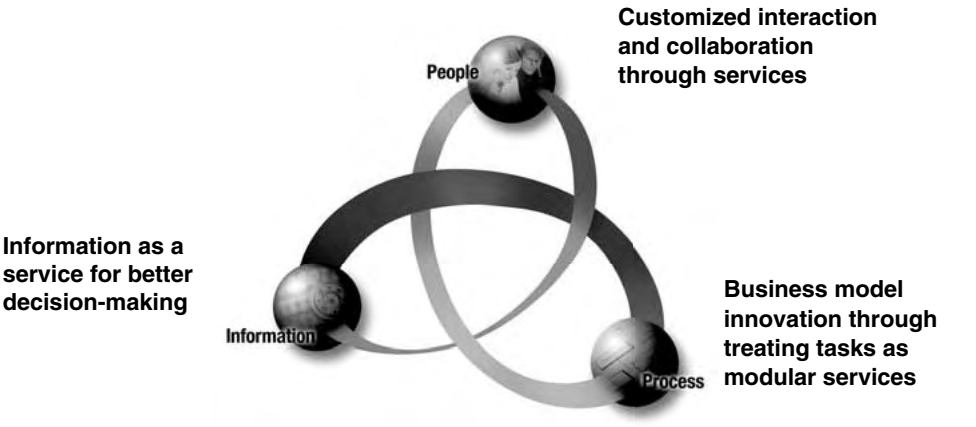



**Figure 11.1** Companies are taking an increasingly business-centric approach to SOA.

## PEOPLE AND COLLABORATION

### CASE STUDY
PACORINI

Pacorini is an international company based in Trieste, Italy. It delivers coffee, metals, foods, and general cargo. The company processes these goods for quality control and schedules them to arrive just when they are needed in the customer's supply chain management (SCM) process. A highly regarded company, Pacorini has 22 locations and 550 full-time employees; it comprises several different companies across three continents and 11 countries. As a market leader in the delivery of green coffee, Pacorini has historically maintained its competitive position by offering timely customer service. Although it used advanced technologies and leading SCM software, the company's internal business processes were not integrated. It was a challenge to manage siloed information and to provide consistent customer service in a 24×7 world. Consequently, Pacorini was concerned about its ability to stay ahead of its competition.

*continues*

Starting with an analysis of its current business processes to define priority tasks and link them using streamlined workflows, Pacorini built a framework of integrated online processes. The company put into place an SOA to construct information retrieval and work processes using repeatable information services customized to fit every task in a consistent manner. The company has implemented an order-enabled portal solution for both internal and external customers. It has also deployed a system-to-system order-management solution with its largest coffee customer in Italy. Pacorini is now in the process of applying the communications standards it developed with its largest customer to nine of its other top ten customers. In the future, it will extend this solution to customers in metals, freight forwarding, and distribution areas. Online ordering will enable the company to automate approximately 30,000 transactions this year, a projected savings equivalent to four full-time employees.

**CASE STUDY**
BUSINESSMART AG

A second example of the people entry point is businessMart. businessMart AG was founded in February 2000 and currently employs a workforce of 28. businessMart conceives and realizes electronic marketplaces and e-business systems for commerce, industry, and handicraft in sectors with catalog-based articles. Measurable improvements and savings are achieved with consistent orientation to the sector processes of its customers and to the in-depth integration of the suppliers and customers' computer systems. The broad spectrum of businessMart's services ranges from conception through technology modules, all the way to the founding of independent, market-leading portal-operating companies. businessMart now carries out the ordering processes of more than 60 suppliers with nearly 3,000 customers and more than 25,000

orders per day. businessMart currently operates two sector portals and additional projects are in preparation.

### Better Integration—but How?

The continuous growth of the portals gives businessMart AG increased transaction revenues and clear growth in subscribers. Accordingly, more outside systems need to constantly be connected to the portal. The decisive head start in technology—the far-reaching integration of the suppliers' and customers' computer systems into the portal—was to be expanded even further for a more economical conversion. businessMart went in search of a solution that would significantly simplify the interface management and provide a reliable, flexible, and easily controllable platform for the exchange of business process information.

### Conversion of the Architecture

businessMart created an SOA and implemented it throughout the entire portal. Within that context, the technology components were connected in independent, individual modules, or "services." Using the modules, business processes no longer needed to be conducted through the bottleneck of a portal center, but instead could be processed in parallel in the allocated modules. The architecture connects the customer systems with the available applications, using a central interface for all the portal components. Using component architecture enables a significantly faster development. The computer systems of new clients can now be integrated just as quickly as separate modules. Efficient and reusable application modules are created, resulting in software maintenance and care that is significantly more economical. In addition, the consistent use of fallback rules ensures that the system stability is not threatened by the failure of a single (outside) component.

*continues*

### The Advantage of the New Solution

The decisive additional value arises for the customers of businessMart AG through the now unrestricted transferability of individual portal services to outside software systems. The most important portal functions can now also be used directly in the customers' usual software via web service interfaces. To call up product details with pictures, exploded diagrams, operating instructions, or even supplier searches, customers no longer need to exit their own merchandise information computer system. These portal services are seamlessly integrated into the software and passed online from the portal. The customers of businessMart profit from faster and more comprehensive possibilities for intervention: Time-consuming, manual information processes were digitized and have thus been made more economical.

For the integration of the customers' various back-end systems, businessMart uses IBM's SOA-enabled software to connect 16 different SAP systems. Now marketplace participants can simplify the flow of information as well as increase their sales and reduce their procurement costs.

In e-business, the contribution margin killers are unclear order positions that generate manual questions by telephone and annoyances through wasting time. This step can now be processed significantly more efficiently through the portal: If the system recognizes an obsolete article number, an unclear entry of a packing unit, or even a format error, the supplier or the customer is contacted in real time. The supplier or customer can immediately remedy the problem directly in the portal through a correction or by creating a conversion rule.

### Well Equipped for the Future

With the transfer of the portal functions to the customers and suppliers' systems, the first step was taken in the expansion of the business model. In the future, companies will no longer exchange

their order information only by means of contacts; they will instead allocate applications and have joint access directly to IT services. A portal will have to take over the role of the interface management to keep the complexity at an acceptable level for the market partner. While in search of a modern technology base, businessMart also found an engine for an evolutionary step.

## PROCESS

**CASE STUDY**
COSCON

COSCON is China's largest shipping container company. As a leader in the shipping and logistics services market, COSCON has 127 container vessels and has shipped more than 320,000 containers to date. Its ships are regularly deployed to ports across the globe, each with its own regulations. To support these diverse requirements, COSCON had an electronic data interchange (EDI) system that consisted of 21 different applications, with a variety of architectures and development languages supported on multiple servers. As COSCON's business continued to grow, its complex IT system hampered the company's ability to respond quickly to its ports of call and its external and internal customers.

To become more competitive, COSCON integrated its existing EDI applications by deploying an SOA. The open standards–based technology approach enabled COSCON to connect its silos of data and software applications to allow its internal business to better interoperate with its customers, partners, and suppliers. This solution leveraged the existing resources within COSCON and augmented it with a solution that improved productivity, allowing for more efficient communication and enabling COSCON to quickly react to changing market conditions.

*continues*

With countries constantly changing their customs requirements, with a change occurring every two to three days and almost one month per change required in the current system, the need for flexibility—becoming flex-pon-sive*—was critical to deployment. Because of these demands, COSCON chose to implement process integration using SOA. The process entry point was chosen to improve the communication between IT and business as well. Some of the processes COSCON chose to focus on were adding ports and reports that the business side needed. By using the process to create business services of the key tasks, COSCON was able to meet government (customs) regulations and to integrate with many applications in different languages. COSCON deployed an SOA approach to consolidate multiple EDI systems and processes.

COSCON has experienced a dramatic increase in internal efficiency and has achieved higher levels of customer satisfaction. COSCON can now respond more quickly to the changing regulations set by foreign ports. In addition, COSCON has reduced the time it takes to configure and modify its IT system, from two to three months to just two to three days. This time savings and greater development efficiency has resulted in higher customer satisfaction levels and has offered sizable cost savings. In addition, it allows COSCON's business personnel to communicate with IT staff and better understand the IT system, and the IT people can also better understand business operations. As we discussed earlier, this alignment of IT and business is crucial for business flexibility.

"Over the past few years, we have witnessed an increase in demand for our shipping services," said Mr. Ma Tao, Deputy General Manager of Information Technology at COSCON. "This increased interest has placed additional pressure on our business, helping us realize that we needed to revamp and invest in our internal technology infrastructure to position our business for future growth."

## CASE STUDY
### Automobile Club of Italy

Whether navigating the crowded streets of Rome or maneuvering the narrow roads that hug Italy's coastline along the Adriatic Sea, drivers count on the Automobile Club of Italy (ACI) to deliver emergency roadside assistance.

As the nationwide provider of roadside services, ACI relies on technical support from ACI Global, which maintains a call center that provides 24×7 assistance. ACI Global has agreements with automotive manufacturers, fleet and car rental agencies, tour operators, banks, and insurance companies to provide multiple products and services via its call center. Center operators handle approximately six million contacts annually, using advanced technologies to provide customers with timely and effective service.

The complete ACI operational network includes 3,000 assistance vehicles, 1,000 operating centers, and 5,000 operators.

ACI Global strives to develop, implement, and maintain value-added services that simplify the operations of its customer companies. The firm had been generating such improvements primarily through continually offering customers new and innovative services that encouraged increasingly rapid response times to roadside emergencies. Unfortunately, isolated business processes and outdated software-design efforts limited ACI Global's ability to redefine its business offerings, frequently delaying the delivery of new products and services.

To satisfy customer expectations for new and innovative services and speedy response times, ACI Global wanted to implement a standardized, flexible design infrastructure that would encourage the rapid creation and delivery of new business functions, in turn streamlining several call center processes and shortening service delivery.

*continues*

ACI Global worked to design and implement an automated call center called "Centrale Operativa" built on an SOA. Now ACI Global staff members can leverage the SOA's open standards capabilities to easily design new support services for customer operators, including automated call-routing systems and improved call tracking and management. The SOA also encourages the reuse of code and processes to further streamline the creation of new services.

ACI Global expects automation and integration to lead to a 20% improvement in customer call response times and a 30% increase in call center operator productivity.

These lessons were learned:

- It was very important up front to involve all the stakeholders.
- Focusing on the business needs made for a smoother production rollout.

## INFORMATION

**CASE STUDY**
PEP BOYS

In 1921, four young neighborhood entrepreneurs in Philadelphia, Pennsylvania, pooled $200 each to start what has become the largest automotive aftermarket retailer in the United States. Today Pep Boys Auto employs more than 22,000 people at its 593 stores in 36 states and Puerto Rico, and reported more than $2.2 billion in sales in 2004. Pep Boys differentiates itself from competitors by being the value alternative to car dealerships, providing exceptional customer service, and being the only retailer that serves all four segments of the automotive aftermarket—do-it-yourself, do-it-for-me, buy-for-resale, and replacement tires.

Pep Boys is leveraging SOA to drive its business goals. In 2003, Pep Boys started to analyze its point of service (POS) and Service Work Order System (for bay service) and realized it did not have the right architecture or applications. The first thing the company did was set up its foundational technical base for SOA with a focus on connectivity and reuse. In this phase of its SOA deployment, Pep Boys leveraged a wide array of existing systems, including IMS/CICS/Old Java. They used a standards-based approach, making approximately 45 calls to back-end systems using web services (WSDL interfaces). They built roughly 200 functional services, with no migration of data required.

For the next phase, Pep Boys extended its deployment to include choreography of several retail processes, including returns and invoicing/billing. They choreographed processes/workflow consisting of 15–20 services. This put the key pieces in place to push new and enhanced functions to its employees in the store. The capabilities enabled by its SOA allowed sales reps to have enhanced, more productive customer interactions. The sales reps were able to turn POS screens around so that they could up-sell and cross-sell using new functionality. At the same time, Pep Boys created and was able to use a single view of the customer for various in-store activities. This is where they focused on the information entry point. The initial pilot was completed in four months at 12 stores, and the total rollout to 590+ stores was completed in April 2005.

Pep Boys started its IT transformation by replacing its outdated POS environment with an IBM Open–POS solution—a next-generation POS configuration built on Java technology–based 360Commerce software running on IBM Store Integration Framework, a specialized instance of an SOA architecture for retail customers, comprising hardware, an operating system, and services from IBM.

*continues*

The business benefits of this SOA entry point of information in combination with other SOA entry points were that Pep Boys experienced faster checkout and increased responsiveness to customer needs, and enhanced employee productivity and efficiency. "Now we can take debit cards, which have a lower fee rate than credit transactions," explains Pep Boys' Bob Berckman, Assistant Vice President.

## CONNECTIVITY AND REUSE

**CASE STUDY**
U.S. Open

The U.S. Open is a tennis event sponsored by the United States Tennis Association (USTA) that is a not-for-profit organization supporting over 665,000 members. It devotes 100% of its proceeds to the advancement of tennis. The USTA leverages SOA to support its business goals and has partnered with IBM since 1990 as its technology supplier. More than 4.5 million online viewers tuned into the United States Tennis Association's (USTA) U.S. Open held in 2006.

The USTA created an integrated scoring system for the U.S. Open. This scoring system helps collect data from all courts and then stores and distributes the information to USOpen.org, the official website of the U.S. Open. The ability to immediately and simultaneously distribute scoring information—with IBM supporting more than 156 million scoring updates for the US Open in 2005—is illustrative of the value and capabilities of a larger technology industry trend known as SOA. The technology supporting the U.S. Open is an example of how SOA can help an organization use its existing computing systems to become more responsive and more closely aligned with the needs of its customers and partners.

For example, umpires officiating at each of the U.S. Open matches hold a device they use to keep score. These devices feed into a

database that holds the collective tournament scores. From there, the constantly changing scoring information is fed to numerous servers that can be accessed through the U.S. Open website. When visitors go to USOpen.org and click the "Live Scores" link, they see the scoreboards for all 18 courts that are updated before the visitors' eyes. This is then used to present visitors with instantly updated scoring information that is presented on the site's On Demand Scoreboards and the "matches in progress" pages.

More specifically, the U.S. Open's scoring system relies on an integration middleware that is a critical part of an SOA. Software acts as an Enterprise Service Bus to transform the messages in-flight from the courts to the devices and to the U.S. Open Web site. A database is also used to support the distribution of the scores and statistics.

Scores and statistics can also be instantly viewed on the Web site and compared with past U.S. Open events and similar competitions. Additionally, IBM technology is helping support the integration of information and statistics related to the tournament, such as individual scores and how they compare with current and past performance of the players and competitors.

When you consider the speed at which these matches are played, you quickly understand how the technology that supports the U.S. Open needs to keep pace as play-by-play results are accurately shared all over the world. The USTA's selection of SOA ensures that fans around the world have a virtual seat to the U.S. Open, with scoring information delivered as it happens on the court.

Linking all of the tournament's information and delivering scores in real time requires a sophisticated information technology (IT) infrastructure that can be easily accessed and understood by USTA subscribers, many of whom are not IT professionals. The USTA is at the beginning stages of an SOA, and the USOpen.org site will be able to accommodate a growing audience of tennis fans worldwide.

*continues*

In fact, nearly 660,000 fans attended the 2005 U.S. Open, making it the world's largest annually attended sporting event. Also, USOpen.org is among the top five most-trafficked sports event Web sites. The site has seen a 62% year-over-year traffic increase, with 4.5 million unique users, 27 million visits, and 79,000 concurrent real-time scoreboards in 2005. Additionally, since SOAs are scalable and flexible, they can easily meet the demands of the constantly changing USOpen.org Web site and the anticipated heavy site traffic produced by 27 million visits—with each visitor spending nearly an hour and a half per visit.

These case studies show that a central element of SOA is the repeatable business tasks that make up processes with modular, interchangeable software so that reuse is possible. Reuse of these services is one of the main drivers of flexibility. In addition, connectivity through an ESB is a key technology that companies need to select for their needs.

## CASE STUDY
### Sprint

In 1899, Cleyson L. Brown sensed the need for a viable alternative to the Bell telephone company and launched the Brown Telephone Company in Abilene, Kansas. In doing so, he began what would become one of the most successful and innovative telecommunications companies in the world: Sprint (www.sprint.com).

After over a century of visionary leadership, Sprint has firmly cemented its reputation in the industry. The company has been first to market with nearly all the telecommunications technologies that inform how Americans communicate today. From the first fiberoptic cable and first digital switch implementations, to the first transatlantic fiberoptic phone call, to the only nationwide Personal Communications Service (PCS) network, Sprint has consistently proven that it is a leader in the telecommunications marketplace.

Sprint now counts over 26 million customers in more than 100 countries, offering them products and services that span traditional phone service, data solutions and Internet services. Sprint's Business Mobility Framework (SBMF) is the most recent extension of the company's innovative heritage and its commitment to providing products and services that help customers find new, more efficient and more profitable ways to do business. The SBMF—a combination of network capabilities, a business approach and a development philosophy—aims to reduce both the cycle times and costs associated with enterprises that want to improve operational efficiency while offering new, mobile-oriented products and services to customers. It enables enterprise developers to extend their applications to mobile workers without having to be experts in mobile technology or the corporate IT environment. As it turns out, the SBMF has opened even more doors than Sprint anticipated.

### Focusing on Core Competencies to Tap New Markets

Sprint is adept at serving its customers. For years, the company has rightfully prided itself on exceptional service delivery and support. What the company's management realized, however, according to Rodney Nelson, senior product manager at Sprint, was that the enterprise market was underserved.

"We realized there was a lot of untapped value," says Nelson. "Within our carrier network, we had a lot of services that people wanted." Services, in this case, meant application functions developed internally that could be extended to other companies for inclusion in their applications.

For example, Sprint developed a locator application in response to recent U.S. 9-1-1 emergency regulations that makes cell phone location information available to emergency personnel, who can then use that information to track people in need of assistance.

*continues*

Sprint realized that this service—the location piece of its application—would be very valuable to customers who manage truck fleets or need to track delivery drivers, or to any company presently using Global Positioning System (GPS) technology.

However, in order to get that value into the hands of the people—the enterprise customers—who could use it, Sprint had to envision and create a gateway whereby enterprise users, regardless of the platforms on which their applications are written, could have access to the service.

Other services include cell phone presence (indicating whether a cell phone is on or off), text messaging and sending voice extensible markup language (Voice XML) messages directly to mobile or wireline phones—integrating voice and data alerts. For example, an airline might broadcast a Voice XML message regarding a delayed flight to everyone on the flight list—to the most appropriate device.

### A Standards-Based Path to Success

Since Web services employ standards, anyone, regardless of their technology environment, can make use of the services Sprint has extended. Literally millions of IT developers can include Sprint services within their applications, and can do so easily.

In most enterprises, at least half of the applications are legacy applications, and the other half typically is made up of commercial off-the-shelf (COTS) applications. To integrate new services into these applications would be time-consuming, complicated and expensive. With Web services and Sprint, however, that manual integration is no longer necessary. Developers need only to integrate the Web Services Description Language (WDSL) into the code and connect to Sprint, where the service is run.

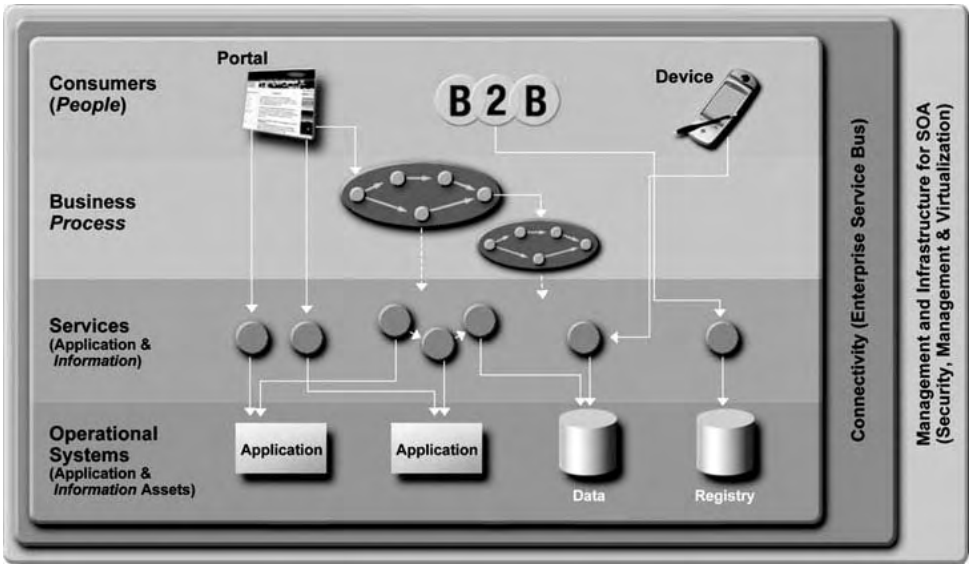
**The Benefits Are Much More Than the Sum of Their Parts**

Sprint has gained numerous new business opportunities, become a leader in the field of mobility workforce enablement and firmly positioned itself to remain a visionary in telecommunications.

Sprint customers that have implemented Sprint's services have seen exceptional cost savings. Developers report that implementation time and effort have been reduced between 40 and 50 percent compared with, most notably, traditional GPS application development. In addition, the time to acquire a location from a cell phone is 30 seconds with Sprint; traditional GPS devices can take up to six minutes. The location information is more accurate, and instead of implementing GPS systems—which can range in cost from US$1,000 to US$3,000—drivers simply need to carry a cell phone, which in some cases is free.

Sprint didn't have to build a whole new IT structure to support this new service. Through its SOA, Sprint just had to unlock and expose capabilities it already had embedded in other processes and applications. This is another great example where SOA is enabling market innovation advantage.

Though in its infancy now, these SOA entry points promise to unleash capability similar to what the Internet—the prior technology evolution of comparable magnitude—already did. Companies employing SOA entry points face more than just technical challenges—there are process challenges and cultural issues, too. In Figure 11.2, you can see an example of how the entry points work in the real world. From the users and consumers at the top, where the services are exposed to people, to the way that processes are broken down into reusable assets made up of application and information components, this picture shows a more powerful, flexible view for companies that can link these pieces together.




Source: SOA Community of Practice, SOA Solution Stack Project

**Figure 11.2** People, business processes, and information sources interact through SOA.

A great way to get started on this flexible IT piece of the equation is to take a self-assessment. In fact, with the assessment at www.ibm.com/soa, you can jointly assess both the business readiness and IT readiness. Answering a set of questions about the business, your technology, and your goals shows your location on a maturity curve. It also suggests projects to begin your enterprise transformation and help you learn the areas before a larger rollout.

You should perform these checklist items:

- Understand what other companies are doing with flexibility and SOA
- Determine how your company can best use an SOA entry point
- Take the SOA assessment to see where your company might begin
- Begin a pilot project to learn the SOA framework

## UNLOCK THE BUSINESS VALUE MULTIPLIER

The next step to SOA value comes when you start to link across the entry points of people, process, and information. This is when you start to realize a Multiplier Effect and your company's SOA business value accelerates. Creating entry points for SOA projects can deliver significant value on their own. People-, process-, and information-centric approaches yield results that can deliver strong ROI. However, the power can be exponential when clients apply SOA capabilities to people, process, and information aspects of a business in combination. We call this the Multiplier Effect and it changes the way you approach SOA.

The Multiplier Effect promises to deliver even greater value to clients by linking people, process, and information through SOA. The promise is that businesses will not only be integrated, but also built for change—built to adapt as market conditions demand greater attention by all parts of the business and shifts in resources. It's no great accomplishment to hard-wire a few databases to a user interface that, in turn, presents information mapped to a particular process. The real value is in creating flexible linkages of all three entry points in a dynamic environment. Clients are continually upgrading and changing processes, applications, databases, and views in the business. Through SOA, all parts of the business—its people, the key processes, and the critical information—can stay linked and supported through that continual change.

In Figure 11.3, begin your view from the center, where you can see the entry points we have been discussing. The companies that have started their journey have seen a higher ROI by combining the entry points of people, process, and information. This increase in flexibility and responsiveness comes from the focus on BPM and composite business services, which are made up of prebuilt, domain-specific modules that form highly customized applications. Composite business applications will become as predominant as the monolithic applications that exist today.

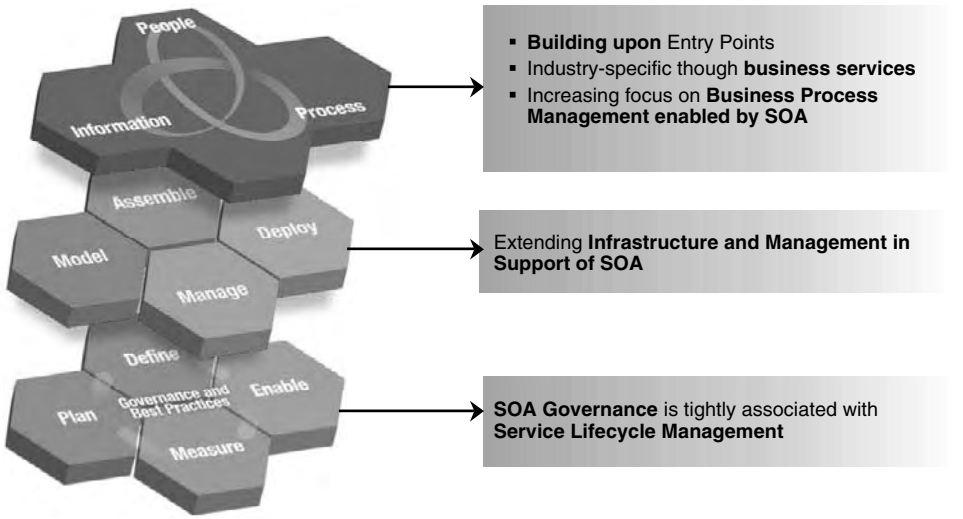



**Figure 11.3** The flex-pon-sive* agenda through SOA

Those same companies, which have seen the value of moving up the stack, also see the power of the right infrastructure. Security, management, and virtualization are all different in a highly flexible SOA domain. These infrastructure services enhance resilience and security to accommodate decentralized services.

Add these items to your checklist:

- After your first SOA project, begin to see the linkages of people, process, and information. As you incorporate an SOA approach to address an immediate business problem, progress on the path to a broader SOA enterprise adoption. These projects generally incorporate reuse and connectivity, and involve information, process, and people (see Figure 11.4).
- BPM is more than a technology—it is a discipline.
- Composite applications will blend with monolithic applications. Check out the SOA business catalog (at www.ibm.com/soa) to see where the future is moving.
- Make sure you evaluate your infrastructure and management capabilities to support your SOA projects.

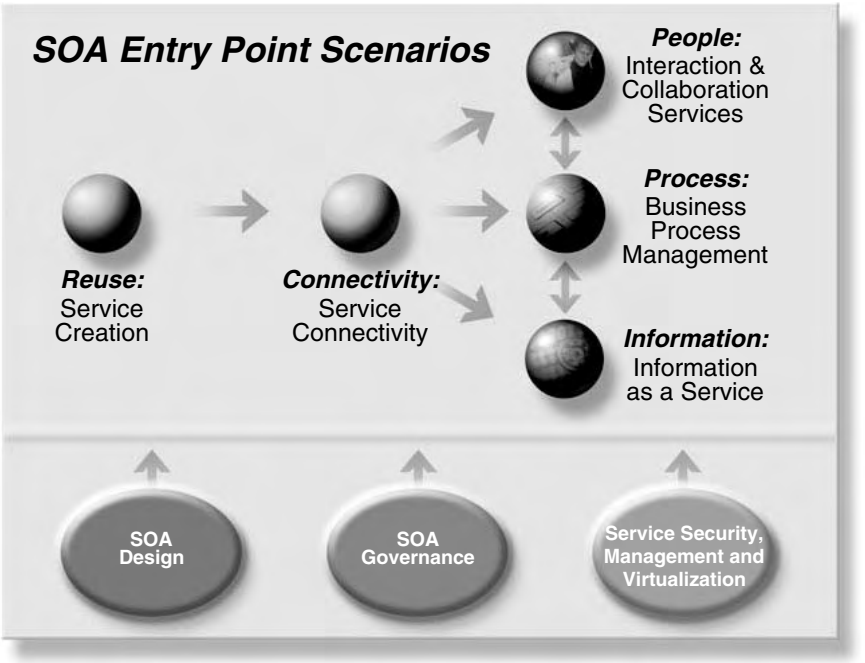



**Figure 11.4** SOA entry point scenarios based on more than 3,000 SOA engagements

For a case study on the overall value of SOA and the impact of combining entry points, s.Oliver is a great example. Working with Sandra Rogers, Program Director, SOA, Web Services, and Integration at IDC, we explored this German company's approach to combining the SOA entry points and its best practices.

## CASE STUDY
### An IDC Case Study: s.Oliver

(From the IDC whitepaper sponsored by IBM, "Service Oriented Architecture as a Business Strategy," doc 204313, November 2006)

#### Background

s.Oliver Bernd Freier GmbH & Co. (s.Oliver) is a multinational retailer of apparel and accessories for men, women, and children. Founded in 1969 in Rottendorf, the company is one of the fastest-growing

*continues*

textile companies in Germany. With more than 2,350 employees, the firm currently operates 49 mega stores under its own management and approximately 240 stores that it runs in conjunction with partners. Its continuously revolving collections are represented in 1,000 branded shops and departments, and distributed through 1,330 stores in over 30 countries throughout Europe. s.Oliver's aggressive growth strategy has the company in line to double its revenues from €820 million in 2005 to reach €1.5 billion by 2010, fueled by further geographic expansion and partnering models.

The fashion industry is very fast-paced as companies such as s.Oliver compete to stay ahead of the curve of consumer preferences. The firm must be able to quickly identify trends and turn over new products, continually introducing new styles and products to stay competitive. Such volatility places heavy demands on the firm's IT environment to be in lock-step with the business, with the latest product information, and subsequently support these offerings throughout all operational aspects of the business, from creation through the entire order and fulfillment process.

The company also must frequently enhance its web presence with features to support what has increasingly become a critical go-to-market and partner-integration channel. With its international reach, s.Oliver must support multiple languages and currencies, yet maintain strong brand control.

### The Business Challenge

When Stefan Beyler, CIO and board member, joined s.Oliver in 2002, he and his team completely reviewed the company's entire systems and application infrastructure with an eye toward innovation. With many corporate divisions and an expansive product portfolio that typically turns over every four weeks, the firm's overall IT approach needed to be addressed in a whole new way. The company's systems environment had to better reflect the overall business strategy; therefore, according to Beyler, s.Oliver's strategy was

not about installing a new application, system, or server. Its strategy was all about speed and agility. The firm must be able to recognize and exploit market trends in real time, apply modern logistics, leverage e-commerce and mobile technologies, and continue to promote a highly collaborative and creative corporate environment.

The company's IT staff of nearly 100 employees is responsible for worldwide operations, with a shared-services model supported by two major data centers, one in Germany and the other in Hong Kong. s.Oliver's IT environment is a heterogeneous mix of many applications that have been acquired over the past few decades, including two major ERP systems and varied database and information sources. Thus, the team needed to manage a tremendous amount of interface logic (estimated at 1,500 data interfaces), and with such a rapidly changing business environment, keeping pace was becoming a daunting and nearly impossible task. The cost of making changes to application integration logic, which required custom coding, was also a widening concern.

Beyler was faced with the challenge of creating an environment that could readily adapt to new business requirements and processes, and manage increasing volumes of information. With corporate expectations of maintaining efficiency and keeping down cost, holistically changing over the company's existing base of applications was not an option. Utilizing SOA to create an on-demand business environment was determined to be the ideal approach that s.Oliver needed, as it would provide the mechanism to address growth and change, yet allow the IT environment to evolve in an incremental fashion with minimal risk, disruption, and expense.

**The SOA Solution**

The company created the s.Oliver Federated Integration Architecture (SOFIA). In September 2005, s.Oliver's IT team began

*continues*

implementing information-centric services to support its highly critical order process and was live in production by the end of February 2006.

One of the critical requirements for s.Oliver's evolving SOA strategy is to utilize technology that can coexist and leverage its heterogeneous application and data resources. It is also important for the company that any new software introduced adhere to open standards for interoperability and investment protection, to minimize future vendor dependencies. The company chose to leverage an IBM suite for BPM enabled by SOA.

Another important part of the s.Oliver IT environment is its use of the people entry point and portal infrastructure to provide access to more than 250 applications and centralized information services, simplifying the user interface and addressing multiple languages the company must support. The portal also enables collaboration and distribution of critical information across the enterprise to facilitate faster processes and decision making.

One of the key business values behind the SOA construct is its inherent flexibility to address change, allowing s.Oliver additional speed to market. This platform allows the IT team to incrementally address new product requirements with minimal impact to upstream applications. The company is recognizing significant cost savings from the reduced efforts of the IT staff to continually maintain hard-wired integration logic.

By applying its SOA strategy, the IT team has thus far been tremendously successful—a sure sign is the volume of requests flowing in from the business. "An interesting point to note," stresses the s.Oliver CIO, "is that the business stakeholders do not see these as SOA projects, nor do they need to have any technical understanding of what a service entails. It is all about providing the business with what it needs."

To facilitate achieving this level of business alignment, normally a business leader is involved with each project to provide that critical link. A team of eight IT professionals is dedicated to the overall SOA strategic agenda. However, for its long-term success, the entire IT community must support the vision and adhere to the reference framework.

Beyler points out the criticality and complexity of outlining and addressing all the processes involved for SOA governance. This includes guidance on how to determine and document requirements, development practices, versioning, monitoring and management, security, and assignment of responsibilities for the many tasks involved in creating and maintaining services. The CIO notes that there is a lot to learn, and it requires a good level of process understanding. The company already had a robust IT governance practice in place; however, it needed to add "SOA thinking" to the equation. To support further automation and SOA governance, the team also anticipates it will leverage a services registry and repository solution in the near future.

**Lessons Learned and Looking Ahead**

According to Beyler, "SOA is a *business* project, not a technology project," and the most significant contributor to its success is addressing the people aspect of the equation. This involves rallying support throughout the entire IT organization, convincing developers through IT operations to cooperate across the many processes and dimensions of SOA design and governance. One activity that Beyler noted to be extremely useful was arranging for IBM SOA training for the s.Oliver IT staff. He noted it was very helpful to have the workshop's agenda address the many activities and roles throughout the IT and SOA lifecycle.

Success, however, involves cooperation and acceptance across the entire business. Many IT organizations look to corporate management to help drive cooperation in the development and use of new

*continues*

technology. However, to Beyler, it is the IT department's ultimate responsibility to drive adoption throughout the company by providing good portal and application features and functions that have an impact on the business.

One of the next technical milestones the company has set its sights on will involve combining operational and nonoperational data within its SOA environment, to support both transactional and data warehouse services. Another will be incorporating service orchestration on top of its Enterprise Service Bus to facilitate functional and process service requirements. Another key business requirement for s.Oliver will be to support offline processing; thus, Beyler and his team will be investigating how to incorporate an SOA-managed client capability.

For s.Oliver, SOA is seen as an enabling competitive differentiation for the company, allowing the company to rapidly introduce new products to market across its many businesses and lines. From a business perspective, the company plans to address functional business processes within its SOA environment to take advantage of the flexibility this architecture enables, including tasks involved in bringing products from design through production, supply chain management, and sales.

## GOVERNANCE IS CRITICAL

If this book has done nothing else but convinced you of the importance of governance as your company moves forward, then that alone is worth its weight in gold. SOA requires an efficient business and technology governance mechanism to make sure that IT efforts meet business needs, and as a means of controlling what services are deployed and how those services are used.

Governance is designed to enable organizations to realize the full potential of flexibility. It addresses issues that, if left unattended, might be inhibitors to gaining the flexibility and time-to-market benefits associated with SOA—essential issues surrounding the

lifecycle of a service. Effective SOA governance is more than just technology. It calls for a lifecycle approach that integrates an organization's people, processes, information, and assets.

In its internal use of SOA, IBM found governance to be that secret to success. "From our point of view, SOA governance is an integral and significant aspect of our overall IT governance, which includes managing process, applications, data, and technology," said Catherine Winter, Team Leader for IBM Enterprise Architecture Governance. IBM started by identifying the appropriate process and roles/responsibilities set up the IBM Architecture Board to govern and manage the SOA environment. This governance strategy helped optimize IT assets across the entire corporation.

Address these keys to effective governance:

- Establishing decision rights for your SOA environment
- Defining appropriate services
- Managing the lifecycle of service assets
- Measuring effectiveness
- Changing the Culture
- Aligning IT and Business

**CASE STUDY**
PEOPLE'S BANK OF CHINA

China's federal bank avoids $1 billion in infrastructure and development costs and eases management of the country's treasury by implementing a nationwide real-time tax and customs payment-collection system based on an SOA.

Owned by the Chinese government, the People's Bank of China (PBC) has been the driving force behind the Chinese commercial banking market since 1949. PBC, which serves as a clearinghouse for the Chinese banking industry, employs approximately 100,000 people at 32 first-line branches, 300 second-line branches, and 2,000 third-line branches.

*continues*

The federally run People's Bank of China (PBC) collects and processes tax and customs payments from all of China's 600 million tax-paying citizens. Historically, the nation's 32 provincial governments would first collect the payments from local banks and then submit the collections to PBC. Delays in this process, as money changed hands from the banks to the provinces and finally to PBC, allowed some provinces to accrue interest on the collections, which complicated management of the national treasury. To simplify and accelerate the process, PBC wanted to collect directly from the local banks. However, such a change would require it to integrate its tax- and customs-processing systems with thousands of different bank systems. The challenge was for PBC to create an efficient exchange system across all of China, without investing massive amounts of time and money in integration-development projects.

PBC can now collect tax and customs payments from the local banks in real time by leveraging a cost-effective SOA. By enabling seamless integration among the disparate banking systems, open standards–based software automatically routes roughly 13 million transactions per day between PBC and the commercial banks, while minimizing the need to hard-code integrations. PBC can efficiently add to or modify the services in the SOA as needed, to accommodate new requirements or implement new functionality with relative ease. PBC has seen business results of an estimated cost-cutting of approximately $1 billion in infrastructure and development costs by taking the SOA approach, and eased management of the national treasury by eliminating processing delays. In addition, it gained business flexibility—flex-pon-siveness*—needed to adapt and improve the exchange system in the future.

To accomplish these goals, PBC built an all-new treasury application infrastructure and SOA that will support more than 800,000 users. Through the SOA environment, they route messages (transactions) to and from the external institutions, handling about 13

million messages per day. To ensure a smooth development process for the system's real-time transaction applications, a team of 20 developers, 10 testers, five analysts, one project manager, and two executives added new processes and development methodologies to the SOA environment. Because of this focus on the Lifecycle of Service Assets through Development and Delivery, PBC will be able to reuse application components to integrate new applications quickly and maintain existing applications easily.

The business value of PBC's SOA environment is that PBC can now easily interface with more than 150 diverse merchant banking, tax, and customs institutions across China, effectively centralizing and standardizing the collection of national treasury information. Using the solution, citizens can submit tax and customs payments online in real time via their bank accounts. Tax preparation that used to require as much as four hours to complete can now be entered and submitted in less than ten minutes.

PBC is able to easily adapt to changing LOB requirements. The integrated environment helps speed the bank's development process and eliminates wasted resources.

In total, the integrated, SOA-enabled system will help PBC save more than $1 billion in national treasury infrastructure, maintenance, and development costs.

## INFRASTRUCTURE AND MANAGEMENT COMPLETE THE PICTURE

To realize the value of SOA initiatives, companies are taking a planned approach to extending existing infrastructure and management capabilities in support of those projects. SOA requires thinking about these areas in a slightly different way. By effectively securing the infrastructure across the people, process, and information boundaries spanned by SOA projects, you can save money, reduce risks, and ensure compliance. Managing efficiently to gain

visibility and control of SOA services and the components underneath them is critical for SOA project success.

By nature, SOA services can be virtualized. Making sure the infrastructure to support services is virtualized allows clients to place and prioritize the services for optimal business performance.

Address these keys to effective governance:

- Establishing the right set of security for your services
- Defining management within the context of SOA
- Determining your virtualization needs to drive performance and the right use of resources

**CASE STUDY**
ING

In 2005, ING became the sixth-largest European financial institution, based upon market value, up six positions from only one year earlier. According to ING executives, this change is a reflection of the company's success in offering innovative and low-cost customer-focused services through a variety of distribution channels, including web services, call centers, intermediaries, and branch offices. However, like many companies out there, ING was facing a growing number of industry regulations and increasing sophistication of its business services. For ING to continue its success, they needed to reduce the time and cost of managing employee access to information while ensuring that staff could quickly respond to business change. Focusing on its entitlement program, ING needed a streamlined approval process that leverages electronic forms and intelligent workflows to enable managers to request and approve authorization requests online. In addition, it was important to their business goals that ING provide a self-service capability so employees could change their passwords without the assistance of help-desk personnel.

The implementation of a common identity management system was a key milestone on the journey to deploying a broader, strategic SOA environment for ING. The removal of security components

from each individual application enabled the implementation of a common, centralized set of security controls. This allows the organization to reduce development and deployment costs, ensures the consistent application of security policy, and provides users with a simple, convenient single-sign-on capability across a wide range of services. The benefit of this decision was a host of business benefits: the total projected savings of €15 million ($20 million) a year, a projected 50% reduction in the number of administrators assigned to support identity management processes within 18 months, an anticipated 25% savings in help-desk administrative costs, the ability to reduce time and cost associated with regulatory reporting, and the ability to reduce turn-on time for new users from 10 days to less than 24 hours.

## SUMMARY

To begin the journey of becoming a flex-pon-sive* company takes both business and IT acumen, with a special focus on your business models and processes. As we discussed in Chapter 1, "The Innovation Imperative," your journey begins with a focus on a real business problem, not SOA. How do you grow? How do you become more responsive? How do you ensure that you have the right skills needed on both the business and IT sides? Start small on your journey and build those needed talents and capabilities; in the long run, SOA will enable your success.

Those who succeed have the longer term in mind—they're flex-pon-sive* in a global world—and they leverage the best practices and learning of other companies. Maximize your company's journey with a shorter time to value:

- Focus on the area that will change the game in your industry.
- Address the area of focus with flexible IT—and SOA—to begin growing your revenue and ensuring flexibility. The SOA entry points are built on business-centric views and are flexible enough to fit your needs. These entry points are more than just hype; they have solid experience woven into the patterns for success.

- Leverage the best practices of others in your industry and those outside. Remember that the new game is the focus on business models and business processes.
- Implement governance, which is critical for cross-business cultural change and a secure, robust infrastructure that is needed to scale and support your SOA projects and undertakings.
- Include a trusted partner such as IBM that provides both the business acumen and advanced technologies to build your journey upon. Because the world is changing, it is not just about technology—it is about the combination of business and IT.

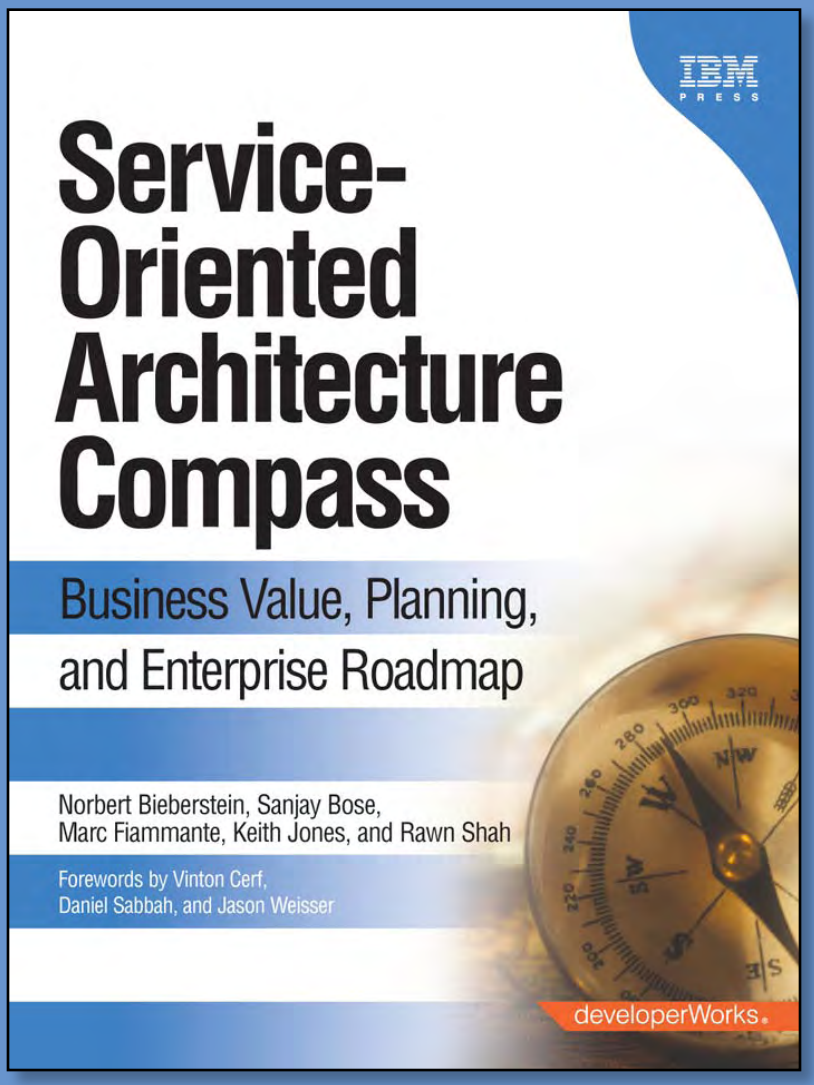
I started this closing chapter discussing the number of shipwrecks off the North Carolina coast due to the unpredictable nature of the forces of nature and man. Today's time seems to be very similar to that N.C. coast. Your success depends on your business's ability to weather unpredictable times and to drive change into the marketplace. It is an adventure that truly takes you to all parts of the world in a global economy. To succeed, similar to those captains of the sea, you will need tools to enable your success.

This book provided you with several of those tools:

- The Component Business Model, to determine which processes to focus on (Chapter 3)
- The business-centric entry points enabled by SOA, to help your business deploy SOA at the rate and pace you need for real business results (Chapter 4)
- A technology roadmap and SOA reference architecture and lifecycle (Chapters 4 and 5, "SOA Key Concepts")
- The business case approach for innovative ideas around process and models, to serve as a framework (Chapter 7, "Three Business-Centric SOA Entry Points")
- More than 30 case studies and examples that can shed light on others' journeys and lessons learned from more than 3,000 real-world businesses (Chapter 9, "The Top 10 Don'ts!"; Chapter 10, "Case Study: IBM"; and throughout)

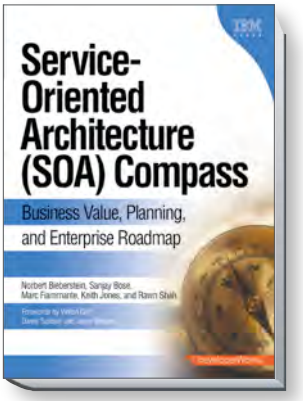
- Maturity models, to determine where you are and what the right approach is for your business (Chapter 7)

Regardless of where you start, your journey toward competitiveness in business model innovation is enabled by the new language of business—SOA and Web 2.0. Becoming a flex-pon-sive* company drives your business to new levels of growth and flexibility in the marketplace. It becomes your language for business success.



Norbert Bieberstein
Sanjay Bose
Marc Fiammante
Dr. Keith Jones
Rawn Shah






# Service-Oriented Architecture (SOA) Compass

## Business Value, Planning, and Enterprise Roadmap

©2008 ISBN: 0-13-187002-5

### Table of Contents



### About the Authors

**NORBERT BIEBERSTEIN,** an IBM solution architect, is responsible for communicating progress with SOA offerings. In total, he has more than 27 years of experience in IT and computer sciences.

**SANJAY BOSE** is the Design Center leader for the IBM Enterprise Integration team. He has more than 12 years of IT industry experience, primarily focused on creating product architecture and design, articulating technical strategy, and designing enterprise application systems using distributed technologies.

**MARC FIAMMANTE** is an IBM Distinguished Engineer with wide experience in large project architecture and software development on multiple environments. Marc has 21 years of experience in IT.

**DR. KEITH JONES** is currently an executive IT architect with IBM in the SOA Advanced Technologies team where he focuses on the definition and implementation of service-oriented architectures with leading-edge customers. He has 30 years experience in the IT industry.

**RAWN SHAH** is the Community Editor (and, formerly, the SOA and Web services Zone Editor) for IBM developerWorks. Rawn has 12 years of experience in the IT industry

IBM Press™

# Chapter 4

# SOA Project Planning Aspects

*"Adventure is just bad planning."*

*—Roald Amundsen*

The architectural consideration of SOA in the preceding chapter offers advice on what directions to choose and how to define the strategic goals for an SOA project. This chapter takes the next step toward execution by focusing on how to plan an SOA project. The topics in this chapter constitute the best practices we have uncovered for forming a project office (see Section 4.1), how to define the phases of SOA adoption, the need for and mechanisms of SOA governance, and finally, the various project roles and how they interact with each other.

This is not intended to be a complete template for a project plan, nor do we intend to show the optimal organizational structure for the parties involved in SOA projects. Based on our vigorous experience with different clients in various industries around the world, we are fully aware that there is no one-size-fits-all solution, nor is there a perfect approach to building an SOA for any scenario. An organization's specific circumstances will dictate its individual needs for project structure and plans. This chapter simply proposes ideas that you can adapt based on your scenario.

The first step is to establish the project office.

## 4.1 Organizing Your SOA Project Office

As seen in the preceding chapters, SOA implies a greater focus on business value. Business models are described as modular processes. This is achieved by breaking down the business

and respective IT systems into components, providing reusability and modularity. Componentization, in this context, applies not just to software systems, but also to the business units across the enterprise and the organization of the enterprise in question. Implementing an SOA project involves not just a consideration of how the project is implemented in an IT infrastructure setting, but in the end, it also results in a business transformation process across the whole enterprise.

To accomplish your project, you first need a roadmap to guide the strategy for your SOA adoption. To build your SOA adoption roadmap, you need to identify who is involved in the SOA project. These individuals should come from the contributing cross-business-unit teams. The actual teams you involve will depend on the level of SOA adoption you choose (see Section 4.2).

Depending on business value analysis and the consequent prioritization of business objectives and services, the team defines "what to do," "how to do it," "who should do it," and "how success is measured." The SOA project team creates the rules, processes, metrics, and organizational structures needed for effective planning, decision-making, steering, and controlling the SOA endeavors. They define the common business service model, the common core processes and business components involved in the SOA project, and the core set of assets that they will use.

Building a suitable team for an SOA project requires a careful avoidance of making radical changes to existing team strategies because such changes can unduly disrupt the culture in the workplace. However, at the same time, the teams need to align with the SOA goals, which usually cut across business units. In addition, the SOA project might need to adopt a management structure—especially at larger IT shops with substantial project goals—to manage the development processes for implementing components or to expose existing applications or legacy functions in terms of the appropriate service granularity.

Achieving the right organizational structure is one of the critical challenges in implementing SOA. At organizations new to SOA, one often encounters strong resistance to change that keeps the focus on short-term successes rather than directing appropriate business transformation to align with the business challenges.

Mature SOA organizations, on the other hand, span business lines and the boundaries of roles while achieving interdisciplinary coordination. However, starting small can help to mitigate risk by allowing you to choose a well-scoped and focused services-integration project that has a modest plan for organizational evolution. A cross-unit, organization structure can address all the aspects of the SOA. Based on our experience, this structure should include the following:

- *SOA business transformation architecture council*: This team is in charge of gathering the business requirements, performing business domain analysis and process engineering analysis, and identifying the necessary business components, services, and process modules. Instead of following a strict top-down approach, the council should use a mixed approach in blending top-down, bottom-up, and goal-based

methods to ensure appropriate services identification. In particular, this team ensures that the exposed granularity of the defined services matches the business requirements and specifications—matching business components to IT components as services. More details on granularity issues and associated services layers are described in Chapter 5, "Aspects of Analysis and Design."

- *SOA technical architecture board*: This team ensures the alignment of business and IT, following industry and enterprise standards, and technically ensures that exposed services match the requirements for evolution and reusability as defined in the general guidelines for the enterprise IT development. Its members are well versed in emerging industry trends, state-of-the-art technologies, and standardization efforts. They are responsible for framing the technical enterprise architecture blueprints (the master IT plan for the enterprise), identifying niche architecture patterns, and promoting reusability principles. They work closely with the SOA transformation team.
- *Component design and development centers*: These are the usual IT teams. They provide design and development of the components and processes, along with new skills such as business process modeling (see Chapter 5). This team delivers a solution design outline, high- and low-level design abstractions, service-oriented analysis and design (the essential aspects of which are described in Chapter 5), and various test phases such as unit, integration, system, and acceptance tests.
- *Operations center*: Finally, there is a production team in charge of the services components operational aspects. These aspects include managing quality of service, enforcing business and service-level agreements, managing the security context, charging back services, and assuring revenue. The team is responsible for rolling out the service, performing regular maintenance, and providing overall system management.

This model for organizing teams is derived and distilled from our experience in projects at midsize to larger enterprises. Often, depending on the maturity level of the IT organization, existing installations can be redefined or transformed to support the SOA projects. After these teams have been identified, you can proceed to creating your adoption roadmap.

Based on their definitions and the associated expert knowledge, each team has a certain scope of decision-making. Depending on the size of the enterprise, the scope, the reach of the project, and the institutionalized IT governance structures, the individuals assigned to the teams can vary. Section 4.3 further explains the need for SOA governance.

## 4.2 SOA Adoption Roadmap

An SOA strategy should not be a big-bang replacement of an existing IT environment; rather, it should be a progressive and evolutionary roadmap. Often an overall replacement is impossible when the majority of people in the IT organization are busy maintaining the running systems. Therefore, the roadmap should reflect an iterative process.

An enterprise has several options for entry points into a service-oriented architecture. These options identify how much the SOA model penetrates into the business and defines levels of adoption. The options are as follows:

- *Initial adoption*: Enterprises that want to reduce risks initially go through a technology validation and a readiness assessment that analyze the technical and business impact in a defined scope. Eventually, the business and technical value realized from this scope can be extrapolated to actual implications for the organization; this usually translates into a deeper commitment to move to SOA. It involves early pilot tests consisting of creating and exposing services from business operations contained in new or existing applications. These tests are used for an early validation of several decision points such as the following:
  - *The capability to transform existing legacy systems*. This might include technical solutions such as messaging, adapters, and connectors, or it might lead to partnership with vendors that can provide products for a service-oriented integration.
  - *The non-functional requirements capabilities* such as performance, security, manageability, and the availability of tooling.
  - *The organizational structure required* to support an evolution of the enterprise, especially one that addresses skills gaps and institutes governance structures.
- *Line-of-business adoption*: At this level, the enterprise will identify a line of business and prioritize processes where the agility and flexibility that SOA offers will increase business value. Of course, the enterprise might have already defined these priorities or have a critical business issue to resolve. In these cases, you still need to assess the SOA applicability to solve the important issue. This involves a broader initial assessment phase and the identification of key metrics and critical success factors.
- *Enterprise adoption*: This level of adoption involves the construction of a business view of a service-oriented enterprise, with a complete prioritization of projects based on business value followed by the architecture and implementation phases. You need to categorize enterprise activities into separate business domains and components that constitute the enterprise. This categorization might already exist within an enterprise or an industry model (for example, the telecommunication eTom model from the TeleManagement forum) that has already-established categories. At this stage, you should establish an SOA governance council with the required empowerment to monitor, define, and authorize changes to services within the enterprise.
- *Enterprise-and-partner-network adoption*: At this level, there is a broad transformation of existing business models or the deployment of new business models involving not only the enterprise, but also its business partners, suppliers, or customers. The enterprise can then select the roles that are appropriate for delivering its value, becoming a service provider, consumer, broker, aggregator, matchmaker, or any combination of those roles.

For each of the prioritized business services and components, the roadmap follows the typical phases of IT project development, with inception, elaboration, implementation, and test and production phases, as typified in the Rational Unified Process™. However, each of these phases includes new activities that relate to the service component identification and realization. Figure 4.1 depicts an overall view of a roadmap, looking at the adoption stages and corresponding activities. This diagram is not exhaustive but gives an indication of potential steps you can follow.

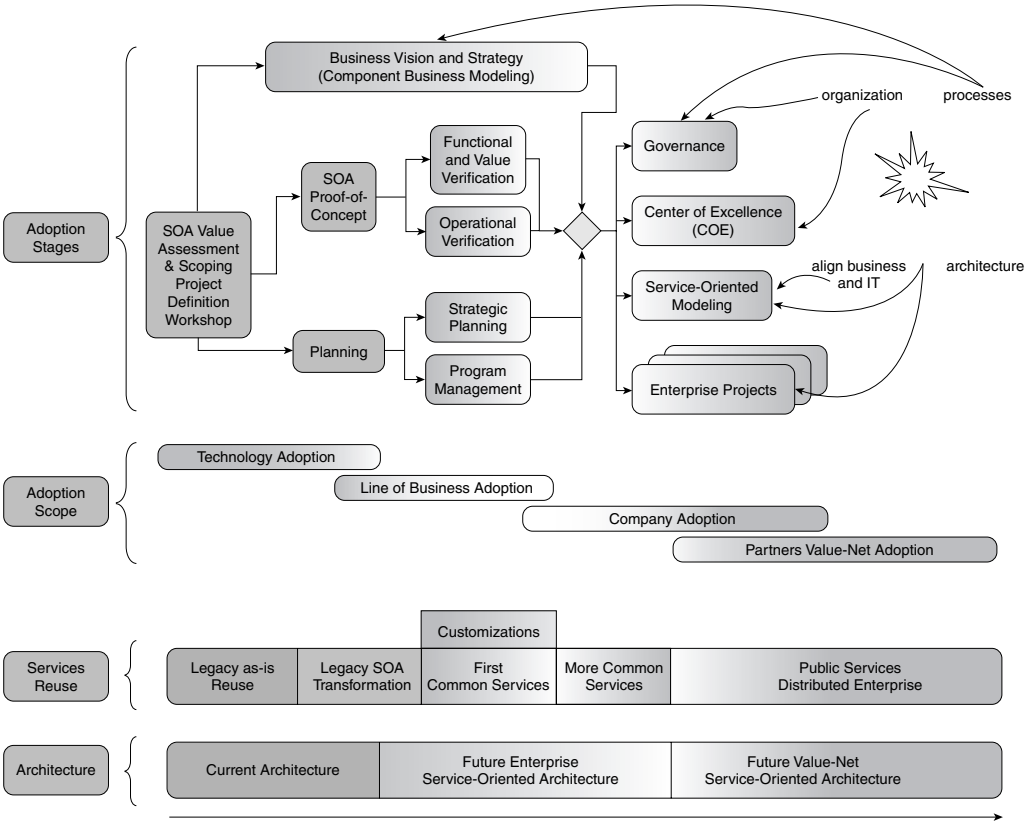



**Figure 4.1** A typical service-oriented architecture roadmap.

## 4.3 The Need for SOA Governance

Enterprises using SOA can adapt to target broader connectivity and increased revenues; on the other hand, doing so requires restructuring applications for greater flexibility and lower costs. This requires the alignment of the business and IT value chain, as described in Chapter 2, "Explaining the Business Value of SOA." With this evolution, the enterprise will also need to adapt the way the business and IT units interlock and define a new way of reflecting business requirements in terms of IT applications. For this reason, organizational governance plays a more prominent role than before. The following sections provide guidance on establishing key governance functions for operating an SOA.

### 4.3.1 SOA Governance Motivation and Objectives

The business operations and the underlying IT infrastructure in an organization must react very quickly to rapidly respond to new business opportunities. Business units have to prioritize new IT services that have to be designed and managed as part of highly integrated and complex enterprise architecture. To achieve this, we discuss in the following sections a set of key governance functions for a successful SOA roadmap.

*Governance* provides an overarching structure to prioritize and then support the enterprise business objectives on a strategic, functional, and operational level. The governance model defines "what to do," "how to do it," "who should do it," and "how it should be measured." It defines the rules, processes, metrics, and organizational constructs needed for effective planning, decision-making, steering, and control of the SOA engagement to meet the enterprise business needs and challenging targets. As previously indicated, the SOA project team is responsible for creating this governance model.

The following are key questions that can help define the appropriate governance structure:

- What business change does the enterprise expect from SOA? Is it a better use of its existing infrastructure at lower costs, does it target new business and interaction models, or does it target both?
- Which roles, responsibilities, structures, and procedures exist to allow business prioritization and IT funding, planning, steering, and decision making?
- How can you develop skills and leadership competency?
- Which principles and guidelines are necessary to optimize the alignment of business and IT?
- What is the appropriate way to structure the business-to-IT relationship while keeping consistency and flexibility to allow the organization to quickly adapt to new changes?
- What is the appropriate level of standardization of services, the service definition, and the description?
- How do you control and measure services and service providers? What key business performance indicators do you need to monitor? Who should monitor, define, and authorize changes to existing services?
- How do you decide on a sourcing strategy for services?

We believe that an accepted and formalized governance model is crucial to successfully achieve business objectives, so we will define important governance functions in the following sections. For fast and high-level acceptance, it is essential to start from the existing enterprise structure and adapt it to the SOA roadmap.

To provide architectural governance, you need an organizational structure to help identify all necessary roles and responsibilities. Based on our experience, it is quite useful to establish an SOA *center of excellence (COE)* to control the SOA roadmap and to support large and complex projects. The COE is responsible for keeping the SOA-based implementation aligned with the business requirements on a strategic, tactical, and operational level. It requires authority over technical artifacts such as architecture blueprints, enterprise templates, and design assets.

### 4.3.2 An SOA Governance Model

A.4.1

In her IBM developerWorks article, Yvonne Balzer describes an SOA governance model on which we based our considerations. SOA governance is an evolution of the ideas of IT governance, introducing a greater business involvement in supporting IT service components. There are different definitions of IT governance, but the IT Governance Institute's definition gives a good general overview:

> ***The IT Governance Institute's Definition of IT Governance***
>
> *IT governance is the responsibility of the board of directors and executive management. It is an integral part of enterprise governance and consists of the leadership and organizational structures and processes to ensure that the organization's IT sustains and extends its strategies and objectives.*
>
> *The purpose of IT governance is to direct IT endeavors to ensure that IT performance meets the business objectives so that the following occurs:*
>
> - *IT alignment with the enterprise results in the promised benefits being realized.*
> - *IT enables the enterprise so that opportunities are exploited and benefits are maximized.*
> - *IT resources are used responsibly.*
> - *IT-related risks are managed appropriately.*

SOA governance incorporates the control of the enterprise model as a set of standardized modular business components and processes, and the prioritization of those based on business value. In summary, the SOA governance model is a combination of organizational structure, joint processes, and relationships that are based on accepted ground rules called *governance principles* and the strategic direction.

### 4.3.3 Strategic Direction and SOA Governance Principles

To sustain the focus on business needs, it is essential to define the strategic direction for developing an SOA. Both business and IT units need a common understanding of the business strategy and objectives. Governance principles and guidelines form the

fundamental basis for any decisions. They shape the solution area and define how business and IT units collaborate. Everyone involved should carefully understand and agree upon these principles, from executive management to individual project personnel.

According to E.G. Nadhan in his EDS Solutions Consulting position paper of April 2003, "SOA Implementation Challenges," there are two main governance approaches:

- *Central governance* is optimized for the enterprise. The governance council has representation from each business domain and from technology subject matter experts. The central governance council reviews the addition or removal of services, as well as changes to existing services, before authorizing their implementations.
- *Distributed governance* is optimized for the distributed teams. Each business unit has control over how it provides the services within its own organization. This requires a functional service domain approach. A central committee can provide guidelines and standards.

Each guiding principle should be defined with a rationale explaining the business reasons and implications. The specific principles for architecture design or service definition, for example, can be derived from these guiding principles. In addition, a common understanding of a structured approach from business to IT is fundamental for defining the architecture. You will find different methodic approaches such as process orientation, business functions, or even component modeling like IBM's Component Business Model approach.

### 4.3.4 Empowerment and Funding

The move to SOA is a paradigm shift driven by the need for more flexible business models, greater integration, and a stronger business and IT alignment. This evolution might face resistance within an organization, which can turn it into just a simpler result of implementing Web services on a small scale rather than a move toward the benefits of a true SOA. In truth, a successful SOA project can happen only with the strong support of senior executives, identified funding, and proper empowerment of the SOA governance body.

One of the pitfalls is the institution of a rubber-stamp governance body or one that has a mere consultative role and cannot enforce its recommendations. At the end of the day, the governance body needs to have proper practical control of project funding.

**Rule of Thumb**

The governance body needs to have proper practical control of project funding.

### 4.3.5 Managing the Risk of an SOA Roadmap

When embarking on an SOA roadmap, the first action of the governance body should be to develop an initial *readiness and risk assessment*. The governance body should then periodically update this assessment during the development lifecycle. Figure 4.2, an

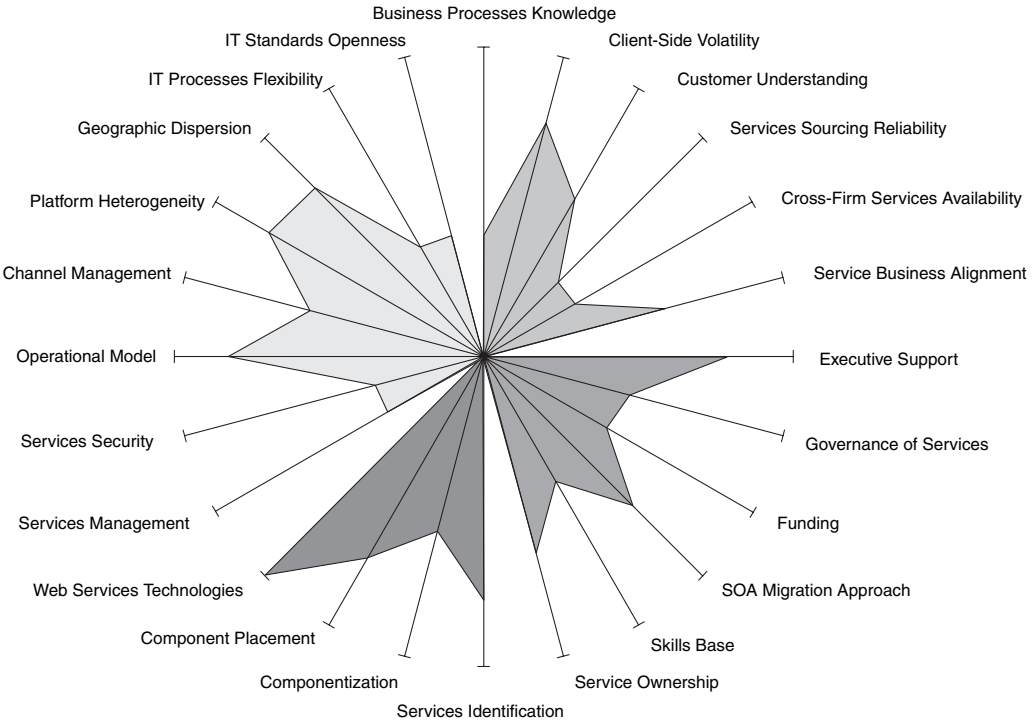



**Figure 4.2** An SOA readiness and risk factor assessment.[1]

example of this assessment, shows important aspects and criteria that need to be taken into account. The scale values and the specific criteria can be chosen based on the situation of the individual project. The goal of this assessment is to identify the business, organizational, and technical gaps and roadblocks between the current state of the enterprise and a future service-oriented business model.

This kind of assessment should balance the vision of the SOA-based solutions with the delivery capabilities of the IT department and should help establish specific a business case for the SOA for the organization. It includes both an evaluation of business readiness and one of IT readiness. It requires customer and partner understanding and determines if changes to the client's or partner's needs can be mapped to existing products or applications in a service-oriented fashion.

1. This was adopted from IBM internal SOA assessment practice and was modified by editors.

The assessment then suggests possible action plans, with focus on improving the less mature aspects of the enterprise relative to the SOA. As before, these improvements to develop the SOA should be executed in well-planned, incremental projects.

### 4.3.6 SOA Governance Processes

Governance processes are those needed for strategic business and IT planning and steering—for example, strategy development, IT technical planning, portfolio management, sourcing, innovation management, and architecture management. Any IT organization also needs processes that provide control. Depending on the size of the organization, these processes should be implemented at the appropriate level matching the size, from individuals to teams to departments or even larger. The following types of processes are essential for successful SOA adoption.

A business component identification and prioritization process:

- Defines a structured approach to model, identify, and prioritize business processes and services components.
- Provides formal definition of the business goals and key performance indicators that can be delivered by the architecture and implementation.

A business exception fallback process:

- Business process models can rarely be exhaustive. No one can preview each and every possibility that can happen in an enterprise. Therefore, there must be rules for exception handling that are set up and agreed to.
- This ensures that the concrete SOA solution architecture has to incorporate entry points that enable certain users or processes to bypass the normal, formalized processes and process exceptions. In a way, this gives another degree of flexibility for ad-hoc business process changes.

An architecture review and approval process:

- Defines a structured approach to review and approve changes to the existing SOA and to make decisions in accordance with the governance guidelines.
- Formal design and service evaluation reviews are key control points of SOA development for the installed governance units.

An architecture exception and appeals process:

- Provides means to appeal architectural decisions.
- Allows exceptions to the SOA architecture to meet unique business needs.

An architecture vitality process:

- Ensures that the SOA is maintained and communicated as new services are incorporated into the architecture.
- Variances to the architecture are documented and communicated.

An architecture communication process:

- Ensures that the SOA is available to all who need access.
- Promotes the understanding of the importance of the SOA.

Having outlined the process we now describe how to launch a governance model in practice.

### 4.3.7 Launching the Governance Model

A.4.1

The process we use to develop a governance model is a three-phased approach (see Figure 4.3). This *governance launch* model was adopted from Yvonne Balzer's developerWorks article "Improve your SOA project plans" and enhanced by the authors. The approach is based on time-constrained SOA engagements. The key to success is to begin to establish the governance functions from day one. To speed up this operation, you can launch the governance model in the following three steps:

Step 1: Operationalize

- Set the governance core functions in place, integrated with the enterprise's business operations.
- Perform the initial SOA assessment.
- Learn and adjust by doing by experiences and available assets, delivering quick results.
- This phase will need experienced practitioners.
- Define the next steps.

Step 2: Professionalize (Automate)

- Build up the necessary structures, processes, methods, and tools.
- Adapt experiences from the operational step.
- Initialize the service-oriented modeling and architecture practice.
- Gather experienced architects and method practitioners.

Step 3: Stabilize

- Teach and train the personnel to run the operation.
- Change from operations mode to coaching mode.
- Need to nurture coaching expertise.

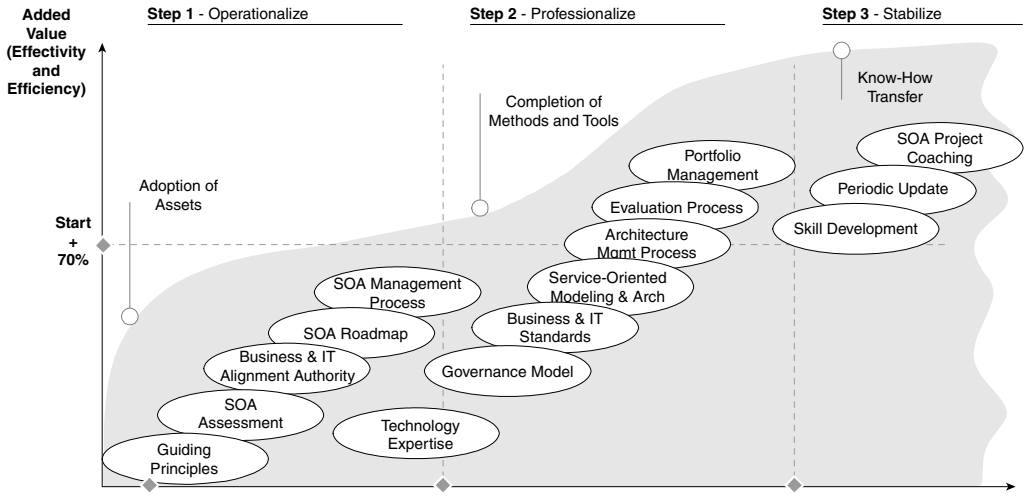



**Figure 4.3** Launching the governance model.

## 4.3.8 Hints and Tips for Success

Even with strong governance, in the real world there are many roadblocks that prevent this type of evolution; thus, it is essential to build on solid ground. The following are some practical lessons we have learned from engagements:

- Set up rules and roles (discussed in Section 4.5) to organize and project-manage the SOA endeavor.
- Communicate regularly. SOA also involves corporate cultural change; therefore, to hurdle barriers, communication is critical, especially between lines of business and technology teams.
- Document each decision, constraint, and assumption to ensure transparency in decision-making and departmental buy-in.
- Define key deliverables and necessary toolsets or templates. These deliverables need to be readable by a variety of parties in the enterprise.
- Set up pragmatic tools for lifecycle management and versioning. Particularly see the discussion on long-lived business processes in Section 4.4.
- Assign a weight factor to each decision and then document and communicate those decisions and their weights.
- Continue to keep a strong sponsorship by all stakeholders and the buy-in of decision-makers.

## 4.4 SOA Technical Governance

With SOA, you can expect that business process cycles will be different from vendor product cycles. As a result, it is inevitable that, in the case of long-running or long-lived processes, you will need to support scenarios in which different versions of a business process exist concurrently on a changing infrastructure. Managing this challenge has implications throughout the project development lifecycle, not just for the runtime but also for the tools and methods used to define business processes within an enterprise.

You can manage the challenge of the dichotomy between business process cycles and product cycle by doing the following:

- Reducing the impact of changes by modularization
- Achieving middleware independence by defining the explicit process state
- Monitoring and handling business exceptions

Each of these topics is discussed in the following sections.

### 4.4.1 Reducing Impact by Modularization

Just as services can have different levels of granularity and permutations in the enterprise, processes also can have such granularity. This granularity appears when processes are designed as a composition of individual process modules. Each module offers a service interface and manages its own particular state internally. It then becomes much easier to change parts of the processes by developing new process modules that are selected from existing services using policies.

### 4.4.2 Achieving Middleware Independence with Explicit Process State

Current business process middleware engines maintain their process state internally. This dependency ties the process instances to the particular middleware engine, sometimes even to a particular version of the middleware. To avoid this, business process designers should elevate the explicit state beyond the engine level at each process step that leads to a waiting state until an external event arrives.

A.4.2

Thus, there is a need to be able to maintain and communicate state as distributed across the SOA. One particular programming model support for capturing these state descriptions is the set of specifications included in the WS-Resource Framework (as published on IBM developerWorks). These specifications allow the programmer to declare and implement the association between a Web service (a process module) and one or more identified, data-typed state components called *WS-Resources*.

### 4.4.3 Business Exceptions Monitoring and Handling

Even if the enterprise has spent a significant amount of time and effort to understand and model its business processes, undoubtedly unplanned business exceptions can still occur.

A fully automated, services-oriented infrastructure that is capable of supporting any such exceptions to the business processes is unrealistic. This means that all business processes and their supporting infrastructure should be designed to allow manual recovery and control. Furthermore, for each business or technical domain, the organization should identify individuals that can handle such exceptions and act on the infrastructure. In most process and services identification modeling activities, the focus is on delivering mainstream models and a few variations. The business analysts must look at making the processes more granular so that unexpected variations and exceptions will be easier to handle in the operational environment.

## 4.5 SOA Project Roles

The notion of handling exceptions in a manual fashion brings to light the key significance of people involved in SOA projects and the roles they might play. SOA projects involve many familiar project roles: project manager, business analyst, architect, developer, security specialist, and system and database administrator. However, these roles were created for different purposes and might have different inherent meanings based on the organization's viewpoint. To appropriately structure an SOA project, you need to consider that these roles might need to evolve to match the componentization and decomposition of the applications into services. In addition, implementing an SOA might also call for some additional roles.

In the following sections, we explain the functions of roles, and then we discuss roles versus skills in a project, followed by the phases in the project where these roles might apply. We then describe how these roles can be extended under SOA, as well as the new roles that follow. Finally, we examine some of the interactions among the various roles in SOA projects.

### 4.5.1 The Function of Roles

There is no single, global, standard definition of all IT jobs, not to mention SOA projects. Many jobs require certifications, a defined knowledge base, or cognitive tests; however, these certifications cannot always truly prove a candidate's applied knowledge, skill transfer, and creativity aspects. Because the team size, the workload, the types of work, and the subjects needed to solve IT problems vary greatly across companies, industries, and geographies, flexible teams with some specialist members are necessary. To coordinate such teams, the leaders (the architects and project managers) have to demonstrate the aforementioned capabilities beyond mere subject knowledge.

In this book, we use the notion of roles because doing so brings some order to the chaos. Roles are related to project phases and define an abstraction layer separate from actual job descriptions and assigned human resources. All project team members take on one or more roles.

*Roles* are a common concept in project management and design methodologies. The role concept establishes a commonly understood vocabulary, which has proven to be a powerful instrument at project-initiation time. A role does not imply that a specific individual person

must execute the tasks that are associated with the role; instead, it implies an identity (an individual, a team, an organization, and so on) that fulfills that part of the process.

### 4.5.2 Roles and Skills

Identifying the roles involved in your project will certainly help, but finding the right people with appropriate skills is crucial. Services typically are technologically lightweight and simple; this is part of their power. With this technology, islands of heterogeneous systems can be more easily opened up for collaboration. However, along with these new possibilities come new sources of errors. SOA projects may be a new kind of project for your organization, but chances are they will not be a simpler kind of project.

An SOA project team setup should reflect these specific issues with the inclusion of appropriate skill levels and talents. We highly recommend a mix of practitioners who have experience with different platforms, different technical problems, and different skill domains. This is especially important for the SOA architect. If such a person is not available to the team, it would be feasible to have additional (part-time) co-architects to fill the gaps.

### 4.5.3 Project Phases

Development projects have different phases, requiring different skills and collaborations throughout the lifecycle, and SOA projects are no different. Independent of your organization's choice of the individual methodology used (waterfall approach or others), most projects will generally include the following development phases:

- Requirements engineering
- Business domain analysis
- Solution architecture outline
- High- and low-level design
- Analysis and design (today mainly OOAD and DB design)
- Various test phases (such as unit, integration, system, and acceptance tests)
- Going live
- Maintenance
- Management

Aspects such as service modeling (for example, using a coarse- or fine-grained interface), choice of SOAP engine (IBM WebSphere SOAP, Apache Axis, or Apache SOAP 2.3), and organizing interoperability tests are primarily examples of Web service–specific considerations. The nature of these issues varies. For example, service modeling prerequisites a different skill and mindset than interoperability testing.

### 4.5.4 Examining and Adapting Roles

The following sections look into a model that shows how existing roles work in SOA development projects. For presentation purposes, we divided the roles in our model into two categories—existing roles and new roles—followed by a discussion on the integration of both categories.

Because SOA projects are just another type of development project, it is not surprising that we find a lot of well-known roles that we can define as a category of *existing roles*. However, some of the existing roles receive additional SOA-related responsibilities. From our SOA project experiences, we saw the need to establish new roles, which are listed as the *new roles* section. Please keep in mind that these are general descriptions of roles rather than detailed job descriptions.

## 4.5.5 A Look at Existing Roles

Let's start with the six roles you may have seen (or participated as) on different technical projects: the project manager, the business analyst, the architect, the developer, the security specialist, and the system and database administrator. Note that this list is certainly nonexclusive, and it is not always applicable to every organization. Therefore, we have limited the list to the most prevalent roles that will later apply to SOA projects.

### 4.5.5.1 The IT Project Manager

The project manager has the overall management and leadership responsibility for the project team. He or she defines and tracks project plans and determines the work breakdown structure. For an SOA project, certain additional skills and knowledge are needed to best perform this role, as described in Section 4.5.5.11, "The SOA Project Manager."

### 4.5.5.2 The Business Analyst

The business analyst harvests the functional requirements of business users and provides domain knowledge to the team. He or she must understand the business language and have industry- and domain-specific skills. In an SOA approach, the business analyst should use a *component business modeling* approach, a technique for modeling an enterprise as disjunctive components in order to identify opportunities for innovation or improvement.

### 4.5.5.3 The Architect

This is the technical leader of the project. The architect's task is to develop the logical and physical layout (structure) of the overall solution and its components.

### 4.5.5.4 The Developer

The developer creates and tests the software implementation. In SOA, the role is not significantly different, with the exception of the code written in SOA projects, which is written as services. This turns the developer into a *service developer.*

### 4.5.5.5 The Security Specialist

The security specialist is responsible for the definition of security guidelines (policies) and the implementation of security means adhering to these guidelines. The domain for this role is described in Chapter 8, "Securing the SOA Environment."

### 4.5.5.6 The System and Database Administrator

This role performs the installation and provides ongoing maintenance on the technical infrastructure: the hardware, operating and database systems, and middleware. Certain

aspects of information integration under SOA were described in Chapter 3, "Architecture Elements." This role typically falls into the domain of the classic database administrator.

#### 4.5.5.7 The Service Deployer

The service deployer takes the development artifacts and installs them in the target runtime environment, generates stubs and skeletons for the target environment from WSDL and installs them together with the service implementations, and provides JAX-RPC (Java API for XML-based remote procedure calls) mapping and handler configuration through services-specific deployment descriptors.

#### 4.5.5.8 The Service Integration Tester

The service integration tester is responsible for the various standard test stages such as integration, load, and acceptance tests. He or she also defines test cases for services interoperability and conformance tests. This role is aligned to the architect and to the governance bodies, as previously described. It is the quality assurance role.

#### 4.5.5.9 The Toolsmith

The role of toolsmith is responsible for designing and implementing the project-specific scripts, generators, and other utilities needed by various aspects of the SOA. The degree of standardization in the Web services world now makes it possible, for example, to develop custom WSDL-, JAX-RPC-, or JSR-109-aware tools.

#### 4.5.5.10 The Knowledge Transfer Facilitator

This role provides access to subject matter experts and technical instructors who bring in extended knowledge regarding SOA, on-demand, and in most cases, Web services concepts and implementation assets. By relating the use cases created during requirements gathering to the functions of this role, this role can easily be taken by so-called *customer-proxies* that represent individual service requests within the SOA.

#### 4.5.5.11 The SOA Project Manager

This role is an evolution of the classic project manager. The SOA project manager not only needs to plan for much shorter delivery cycles, but he or she must also establish new acceptance models. The project manager has to work with service providers to establish the appropriate service-level agreements and resource usage. This role becomes more important with increased use of aggregated services (those that are composed of other services).

#### 4.5.5.12 The SOA System Administrator

In this role, in addition to managing and monitoring the platform infrastructure, the system administrator also manages the business and service-level agreements within the SOA.

### 4.5.6 A Look at New Roles

In addition to the revised roles, there are other roles involved with SOA projects: the SOA architect, the service modeler or designer, the process flow designer, the service developer,

the integration specialist, the interoperability tester, the UDDI administrator, the UDDI designer, and finally, the services governor.

#### 4.5.6.1 The SOA Architect

The architect role evolves into an *SOA architect*, a mediator between business and technology. More than a structural architect, as in the role of a classic software architect, the SOA architect acts more like a city planner, with a higher-level view. This person plays a dynamic business-to-IT adaptation role to transform the ideas and concepts of business operations into terms and concepts available in the IT infrastructure. In this role, the SOA architect is responsible for the end-to-end service requestor and provider design and takes care of inquiring about and stating the non-functional service requirements.

#### 4.5.6.2 The Service Modeler or Designer

A service modeler applies data and function modeling techniques to define the service interface contracts, including the schemas for messages during an exchange. The service modeler works with the SOA architect to create these contracts.

#### 4.5.6.3 The Process Flow Designer

In place of an integration specialist, this role investigates the explicit, declarative, service orchestration (aggregation, composition) possibilities. It concentrates on the technical process flows that support given business processes. It is an optional role in most cases.

#### 4.5.6.4 The Service Developer

The service developer is typically an enterprise developer, expert in programming models such as J2EE and familiar with Web services concepts and XML. The role develops service interfaces and implementations on the provider side and service invocation code on the requestor side of service interactions. The *service developer* is probably the best-equipped role today. An SOA development environment focuses on providing tool-based support for designing and deploying service-based components.

It is the role of the SOA architect to ensure that the SOA developers are not overemphasizing the use of technology. It often becomes easy to expose any piece of logic as a service but at the cost of ignoring best practices for defining the granularity of these services, thus impacting overall performance and manageability. For example, you can easily turn a useful, simple, calculation function into a service so that it can be called from numerous other services.

A service developer using WS-* and J2EE standards-based tooling might also have some subroles:

- *Web Services designer and programmer*, with implementation skills in J2EE, C++, or .NET
- *Legacy integration and adaptation designer*, with services-enabled adaptation skills

#### 4.5.6.5 The Integration Specialist

Integration specialists are mediators and users of both the service modeler's and the process flow designer's work. They typically have a broad-based technical knowledge in the integration field because they will need some understanding of SOA systems, enterprise integration means, business processes, and applications coded in Java or other languages. Tools like WebSphere Business Integration Workbench™ allow the person in this role to compose complex systems of available services.

#### 4.5.6.6 The Interoperability Tester

The interoperability tester verifies that any developed requestor and provider implementations will interoperate seamlessly. Another primary activity is to ensure Web Services Interoperability (WS-I) conformance and industry standard conformance.

#### 4.5.6.7 The UDDI Administrator

The UDDI administrator defines how a generic UDDI data model can be customized and populated with services. This is, in most cases, an optional role. It also depends on whether UDDI is chosen as the standard or there is an alternate proprietary data model in the organization. In this case, there might be another similar role, often part of the IT governance body, that acts as administrator.

#### 4.5.6.8 The UDDI Designer

There might be the need for a *UDDI designer* who is responsible for planning, designing, and building the UDDI registry where services are published and announced for the various levels of the SOA.

#### 4.5.6.9 The Services Governor

This is an important new role that has the cross-enterprise responsibility, on the SOA governance team, to validate and select the business services most appropriate for the given enterprise and then identify who owns them. This role can be played by either SOA architects or business analysts when working in the SOA governance team.

### 4.5.7 Integrating Existing and New Roles

These new roles originate from existing ones (for example, SOA architect and service developer). However, we believe that for the new roles, the introduction of new names for these roles is justified. Each of the roles addresses a different aspect of the project as a whole. Earlier we stated that one person typically wears several hats, in other words acts in more than one role. However, a project's risk decreases if different people with broad and diverse skills are on board. There are situations in which only such a purposeful cooperation of different individuals can unveil the crucial issues of the project and lead to a sound solution.

On the other hand, the communication overhead increases with each new team member. There is no single or simple answer to the roles-to-people mapping challenge. There are

many different opinions and controversies regarding how it should be approached. As previously mentioned, the assignment of individuals to these various roles depends on the situation, the skills and experiences available, and the scope of the project.

Rather than entering a debate about what is the optimal formation, consider the following scenario: A fictitious company in the insurance industry has decided to build a new set of mid-office business applications for risk and policy management, and the set has to interface with two different backend systems. Both backend systems have been built as J2EE applications: one uses EJBs and the other uses only servlets, JSPs, and JDBC to connect to its customer and contract databases.

During the initial stages of the development project, the roles defined in the example are assigned to team members. In addition to the Web services–specific activities, the standard project tasks and roles are also identified and assigned. To illustrate how a team setup can look, Table 4.1 shows an example of selected new roles, their tasks, prerequisite skills, and supporting tools. Additionally, we list the other roles on the team that collaborate with each selected role.

Because the definition of roles depends on the concrete project situation, assignments to the actual people cannot be prescribed in a standardized way; they have to be defined and assigned per the needs of the situation. Assignments can be used by the project manager in cooperation with the SOA architect to create the work breakdown for the project at hand. Further corporate guidelines, restrictions, and best practices are influencing the final project setup and management.

A.4.3

In addition to the previously mentioned roles at the SOA level, there is a complementary list of roles specific to Web services in the developerWorks article, "Web Services project roles," by Olaf Zimmermann and Frank Mueller (2004).

## 4.6 Summary

After the SOA governance model has been launched, it sets the stage for the services modeling and architecting phase, which also includes the design of the necessary supporting IT infrastructure. This step requires the use of a formalized method. The next chapter covers the best practices and the modeling and architecture method we have captured from many of our SOA projects.

**Table 4.1** The Responsibility of Selected New SOA Roles

| Project Role | Performed Tasks | Collaborates With | Prerequisite Skills | Supporting Tools |
|---|---|---|---|---|
| SOA architect | Solution outline, requirements analysis, architectural decisions, component modeling, operational modeling, communicating business issues and components to services | Any other team member plus line-of-business (LoB) representatives | General IT architectures, J2EE technology, XML, XML schema Web services and SOA concepts, platforms, best practices, business knowledge | UML editors, office suites |
| Service modeler | Interface contract design, WSDL editing (top-down, bottom-up, meet-in-the-middle) | Business analyst, SOA architect, service developer | WSDL XML schema and namespaces J2EE technology | WSDL editors, Java-to-WSDL generators |
| Process flow designer | Business process modeling, assembly of atomic services into chains (processes) | Service modeler, business analyst, SOA architect, LoB representatives | BPEL4WS, WSDL | Graphical flow modeling tools, BPEL4WS generators, corresponding runtime support |
| Service developer | Service provider coding, service requestor coding, provide SOAP header handlers if needed, code documentation | Service provider coding, service requestor coding, provide SOAP header handlers if needed, code documentation | J2EE, XML, SOAP, WSDL | Web services wizards in IDEs, WSDL-to-Java generators |
| Interoperability tester | WSDL inspection, SOAP envelope tracing, conformance testing, troubleshooting | Service developers (requestor and provider side) | SOAP, WSDL, WS-I profiles | TCP/IP tunnels and monitors, WSI test tools |

## 4.7 Links to developerWorks

**A.4.1** Balzer, Y. *Improve your SOA project plans*. IBM developerWorks, July 2004. http://www-106.ibm.com/developerworks/webservices/library/ws-improvesoa/.

**A.4.2** IBM. *The WS-Resource Framework*. IBM developerWorks, March 2005. http://www-128.ibm.com/developerworks/webservices/library/ws-resource/ws-wsrfpaper.html.

**A.4.3** Zimmermann, O. and Mueller, F. *Web services project roles*. IBM developerWorks, June 2004. http://www-106.ibm.com/developerworks/webservices/library/ws-roles/.

## 4.8 References

Belbin, R. Meredith *Management Teams, 2nd Edition*. Elsevier Books, 2004.

Belbin, R. Meredith *Managing without Power*. Butterworth-Heinemann, 2001.

Bieberstein, N. *Application Development as an Engineering Discipline: Revolution or Evolution?* IBM Systems Journal, Vol. 36, 1-1997.

Hess, H. M. *Aligning technology and business: Applying patterns for legacy transformation*. IBM Systems Journal, Vol. 44, 1-2005. http://www.research.ibm.com/journal/sj44-1.html.

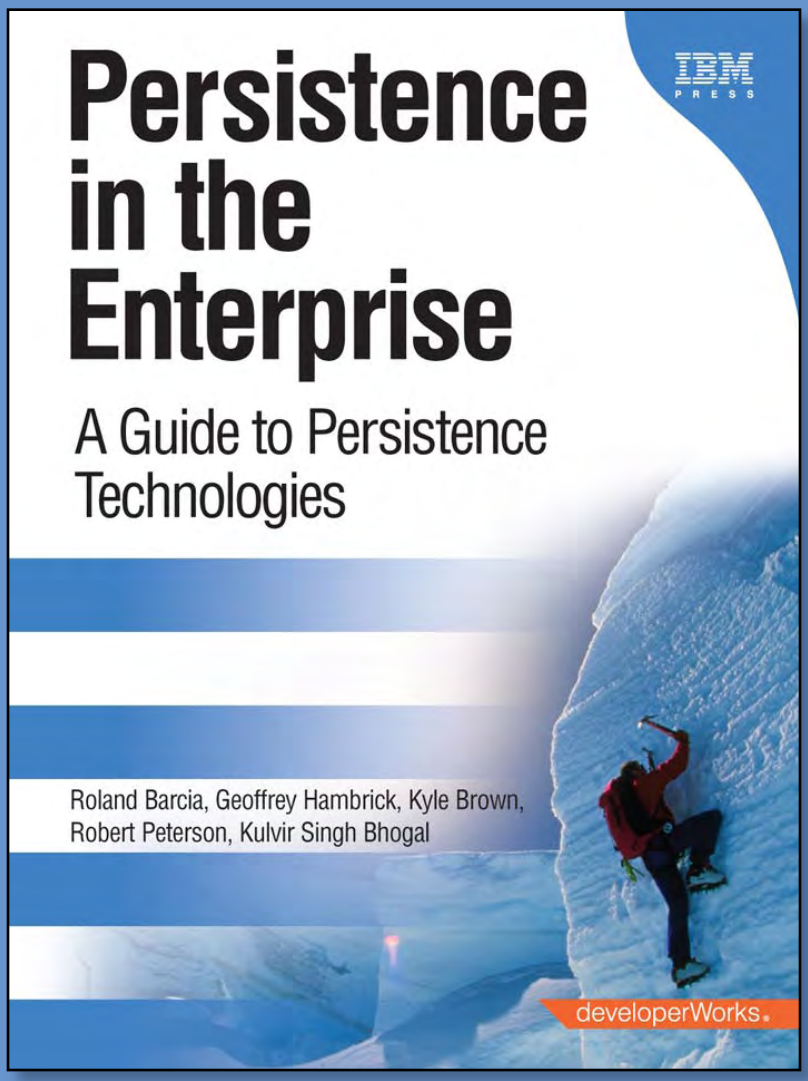
IT Governance Institute. *IT Governance definition*. http://www.itgi.org/template_ITGI.cfm?Section=Purpose&Template=/ContentManagement/HTMLDisplay.cfm&ContentID=14697.

MacMillan, K. and Downing, S. *Governance and Performance Goodwill Hunting in Strategy Dynamics*, Henley Management College, 2001.

Meredith, Jack R. and Mantel Jr., Samuel J. *Project Management—A Managerial Approach, 3rd Edition*. John Wiley & Sons, 1995.

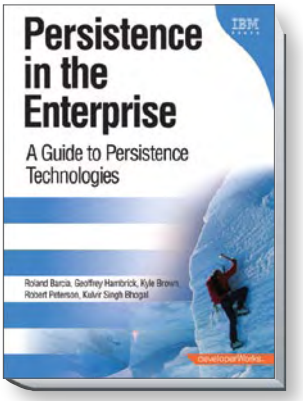
Nadhan, E.D. *Service Oriented Architecture Implementation Challenges*. EDS Solutions Consulting, position paper, April 2003. http://www.eds.com/services/innovation/downloads/so_architecture.pdf.

Telemanagement Forum. *Homepage*. http://www.tmforum.org.



Roland Barcia
Geoffrey Hambrick
Kyle Brown
Robert Peterson
Kulvir Singh Bhogal






# Persistence in the Enterprise

## A Guide to Persistence Technologies

### Table of Contents



©2008 ISBN: 0-13-158756-0

### About the Authors

**ROLAND BARCIA** is a Senior Technical Staff Member (STSM) and Lead Web 2.0 Architect within IBM Software Services for WebSphere®.

**GEOFFREY HAMBRICK** is a Distinguished Engineer in the IBM Software Services for WebSphere Enablement Team.

**KYLE BROWN** is a Distinguished Engineer with IBM Software Services and Support SOA and Emerging Technologies Team.

**ROBERT PETERSON** is a Senior Consultant for the Enablement Team within IBM Software Services for WebSphere.

**KULVIR SINGH BHOGAL** is a Senior Consultant with IBM Software Services for WebSphere SOA Delivery Team.

IBM Press™

# Chapter 4

# Evaluating Your Options

Now that you have a solid understanding of the central role that persistence plays in enterprise architecture and a history of how Java persistence mechanisms and related frameworks have evolved, your next step is to decide which one to use. For purposes of this book, the options you have to consider include Java Database Connectivity (JDBC), iBATIS, Hibernate, Open Java Persistence API (JPA), and pureQuery (a Language Integrated Query for Java). Chapter 1, "A Brief History of Object-Relational Mapping" provides the background on why these five options were chosen. In a nutshell, they represent a good baseline and cross section of the approaches to ORM used by the persistence mechanisms in common use when this book went to press.

This chapter will take you through some best practices for conducting an evaluation based on an objective questionnaire that we will use as a method and template for comparison in the following chapters.

## Comparing Apples to Apples

Many people might take the adage about always comparing apples to apples literally, and simply start evaluating each of these five persistence mechanisms against each other to determine which one is best. This "exhaustive" approach to comparison is shown graphically in Figure 4.1.

One problem with an exhaustive comparison is that it is relatively expensive. You would have to make 10 separate comparisons to rank the 5 mechanisms with respect to one another. If you decide to consider another persistence framework, such as SQL for Java (also called SQLJ), which has many excellent articles on how to use it [A.4.1], you would have to evaluate 5 more mechanisms for a total of 15. Consider another on top of that, such as TopLink JPA (see its website for details [Toplink]), and you have to compare 6 more for a


A.4.1

total of 21. The relative effort of the evaluation increases according to the general formula: n(n -1) / 2, where "n" is the number of mechanisms being evaluated. This formula pegs an exhaustive evaluation as an $O(n^2)$ solution—an approach to be avoided if there is any way around it.

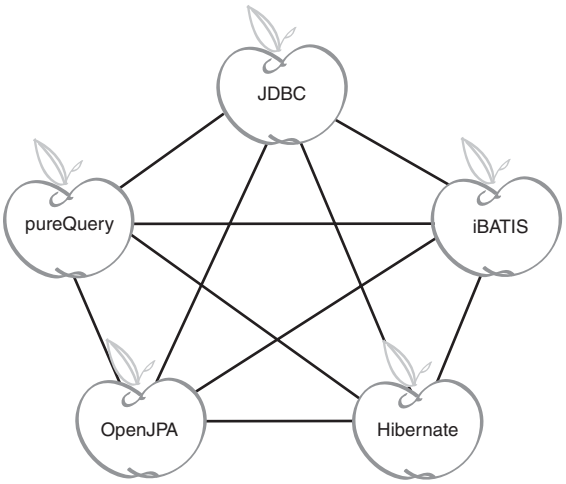



**Figure 4.1** An exhaustive apples-to-apples comparison among persistence frameworks.

For those not familiar with the "O" notation, it is a formal way to describe the "order" of computational complexity, and is useful for comparing solutions (as we are doing now with respect to evaluation approaches).

## Putting Good, Better, and Best into Context

But this analysis of complexity is somewhat putting the cart before the horse. Labels like "better" and "best" have meaning only within a specific context of measurement and interpretation of the resulting values. This context determines the specifics of how to compare two or more frameworks against each other.

For example, you could simply compare the number of Java methods associated with each API that you can glean from documentation on the assumption that the one with more functions is better.

We do not recommend this approach because it leaves you at the mercy of misleading marketing materials or incomplete documentation. Also, it is not clear that more methods make a framework better. However much people may disagree about the value of a given measurement or its interpretation, within that context there is no argument about which framework is better or best—and therein lies the value of objectivity.

## Establish an Independent Standard

The pentagon formed in the center of Figure 4.1 hints at much better context for comparison—if you consider it to represent the union of the features supported by the five options. You can think of each feature in the union as implying a "yes/no" question in a checklist, such as this:

*Does the framework support <feature X>?*

A union of features is better because you are measuring against a relatively independent standard. An independent standard enables us to establish what an ideal framework should support, and compare what each mechanism actually supports against that ideal.

A helpful side effect is that an evaluation against a standard like a union of features is actually simpler than an exhaustive one—because you do not compare the features of every framework with every other. Instead, you merely compare the features of each mechanism to the checklist (making this an O(n) solution). And the comparison is still among "apples" if you consider the superset of features in the union as the "best" possible example of a persistence mechanism (sometimes called an *archetype*), as shown in Figure 4.2.

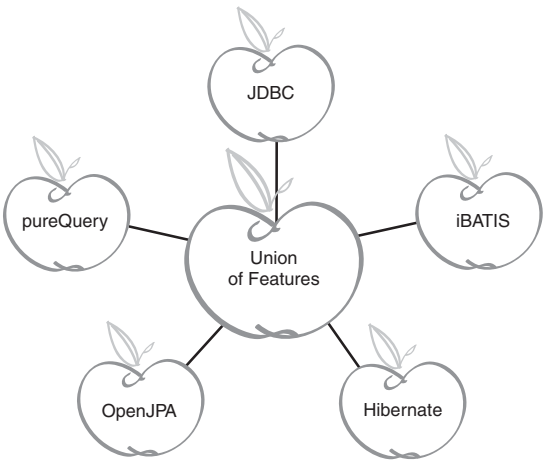



**Figure 4.2** Using a union of features as an independent standard for comparison.

Developing a union of features is pretty straightforward. First pick an arbitrary framework and use its features as the starting point for the union. Then examine each other mechanism in turn, enumerating its features and adding any not already found in the current set.

For example, here are some introductory paragraphs about JDBC we found by browsing Wikipedia:

> **JDBC** is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

> The Java Platform, Standard Edition, includes the JDBC API together with an ODBC implementation of the API enabling connections to any relational database that supports ODBC. This driver is native code and not Java, and is closed source.

This cursory description leads to some good questions to ask about every framework that you might not have thought to consider, such as these:

*Is the framework part of Java Platform Standard Edition (JSE)?*

*Is there an open-source implementation?*

Deeper digging into the materials shows that JDBC supports invoking arbitrary SQL statements and batching multiple statements into one request—providing full functionality and efficiency. Therefore, you should consider asking the following questions as well:

*Does the framework support invoking arbitrary SQL statements?*

*Does the framework support batching multiple statements?*

Turning next to iBATIS, its Wikipedia article begins with the following:

> **iBATIS** is a persistence framework which enables mapping SQL queries to POJOs (Plain Old Java Objects). The SQL queries are decoupled from an application by putting the queries in XML files. Mapping of the retrieved objects is automatic or semi-automatic.

These features are important for ease of use and maintainability, and lead to even more questions to think about:

*Does the framework support automatic mapping of query data to Java objects?*

*Does the framework support modifying queries independent of the Java code?*

Hibernate, pureQuery, and JPA each add their own features to the union as well; but as you examine each framework in turn, you add fewer and fewer (because most of the features have already been encountered).

## Make a List and Check It Twice

After you complete the union from the initial scan of each framework, the next step is to examine each again and determine whether it supports each of the features in the checklist. This second scan is important because it may be that a given mechanism supports a feature but never explicitly documented that fact. The total number of features supported usually increases. At the same time, you find out how many features a framework does *not* support.

It is possible to compare as you go while developing the union as long as you recheck *each* previously evaluated mechanism whenever a new feature is added. However, this approach is functionally equivalent to a second scan.

To illustrate, Table 4.1 summarizes the evaluation against the checklist questions we discussed so far.

**Table 4.1** Results of the Evaluation Against the Checklist of Features Derived from the Union

| | JDBC | iBATIS | Hibernate Core | OpenJPA | pureQuery |
|---|---|---|---|---|---|
| Part of Java SE or EE Standard | Y | | | Y | |
| Open Source | | Y | Y | Y | |
| Arbitrary SQL | Y | Y | Y | Y | Y |
| Batching | Y | Y | Y | Y | Y |
| ORM | | Y | Y | Y | Y |
| External Queries | | Y | Y | Y | Y |
| | | | | | |
| **Total supported** | 3 | 5 | 5 | 6 | 4 |
| **Not supported** | 3 | 1 | 1 | 0 | 2 |
| **Percent** | 50 | 83 | 83 | 100 | 66 |

It is reasonable to consider the framework with the most features supported, or the fewest not supported, or the largest percentage as "the best" within this context. Any of these three measurements used to summarize the table would serve as a more meaningful context for comparison than a simple count of advertised features.

## Keep It Real

Comparing to a union of features is not likely to be a satisfactory basis for your choice of mechanism because you are still at the mercy of what the framework documentation has to say about which features are important. Even though you have a collective group of opinions to consider, the basis for comparison is not truly independent.

An even more meaningful context for comparison is to create a list of your project's requirements with respect to persistence features and then evaluate each framework against that checklist. This focus on your project requirements also ensures that the label "best" has a *practical* meaning—it is applied to the mechanism that does the best job of meeting your needs.

One benefit is that even less effort is required than that to compare against a union of features. You do not need to exhaustively research all the frameworks first to develop the

union; you simply determine whether a given framework has the required features—which are likely to be a much smaller subset than the complete union, but still another form of "apple," as Figure 4.3 shows.

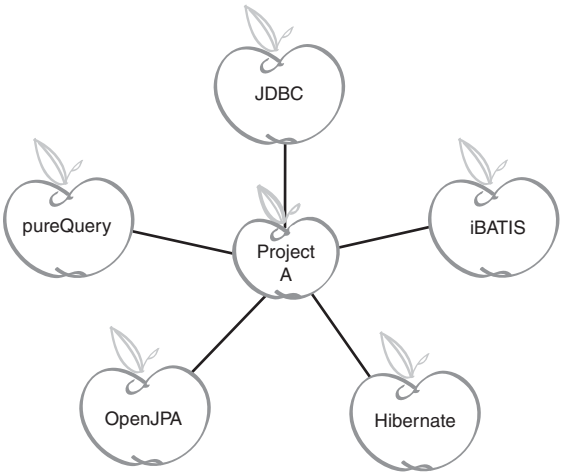



**Figure 4.3** Using your project requirements as a more practical standard for comparison.

Focusing on your immediate needs can further simplify an evaluation because you can stop considering an option that fails to meet a "must have" requirement. For example, project "A" may require a robust ORM that allows you to reject JDBC as the mechanism early in the evaluation process.

A more complete approach would be to weight each requirement based on its importance to your project, and then choose the framework with the highest score.

However, this continual comparison to project requirements, even though they are a smaller list than the union, is reminiscent of an exhaustive comparison (Figure 4.4 illustrates the similarity to Figure 4.1). In fact, the long-term costs are likely to be more than for an exhaustive comparison.

But there are other issues with this "every man for himself (or herself)" approach to evaluation. One is that you are spending time evaluating frameworks when you should be implementing applications. Another is that an inexperienced team may not know the right questions to ask. And still another is that you may end up with inconsistent implementations across the enterprise even when the projects are similar.

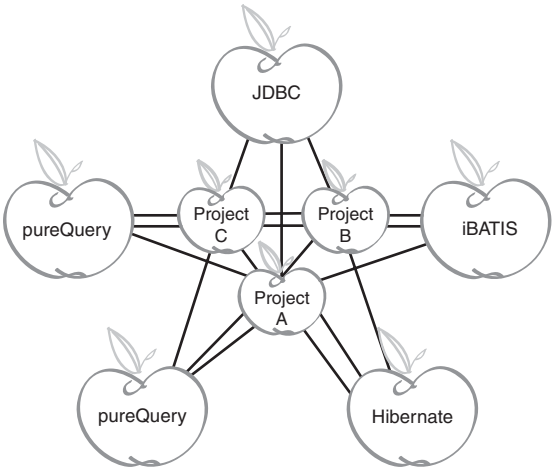



**Figure 4.4** Long-term cost of using individual project requirements as a context for comparison.

## Persistence in Your Enterprise

The word "enterprise" suggests a solution. Just as it is better to develop an independent standard from a union of advertised features and compare against that, it is better to develop a comprehensive checklist of enterprise requirements from a union of those features needed in previous (or proposed) projects.

Given our previous examples, project "A" would contribute ORM support as an enterprise requirement, project "B" would add the need for dynamic SQL, and project "C" would need both.

Ultimately, your enterprise requirements go far beyond just the technical features usually described in the framework documentation, and include aspects such as business drivers and functional and nonfunctional requirements that we discussed in previous chapters. These aspects lead to additional questions that you should consider in your enterprise wide checklist, such as the following:

*Are there referenceable projects that have successfully used this framework?*

*Is there a large pool of practitioners skilled in using this framework?*

Another benefit of this broader focus is that only one evaluation of the frameworks against the enterprise requirements is needed to support making your choices later. Your checklist of enterprise requirements becomes the ideal superset of features to compare the frameworks against—still keeping it "apples to apples," as shown in Figure 4.5.

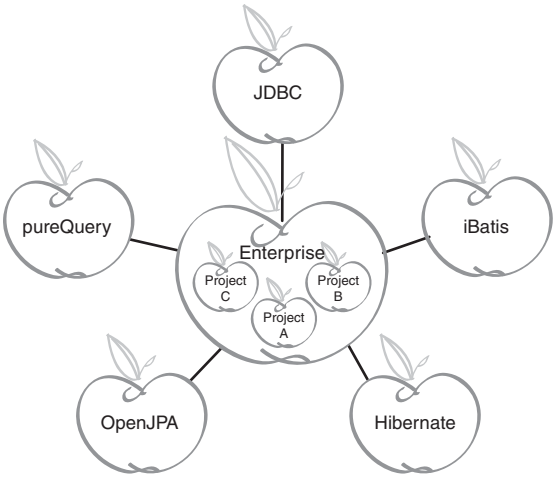



**Figure 4.5** Using your enterprise requirements as a standard to drive a single evaluation.

## One Size Does Not Fit All

However, you should consider the possibility that you need to choose two or even more frameworks for use within the enterprise—not because they end up with the same objective score in an evaluation, but because there are important underlying differences in the requirements of different *types* of projects.

So when you develop your checklist of enterprise requirements, create a weighted subset for each kind of project that you are likely to encounter, and optimize the choice of framework for each type.

For example, it is likely that highly tuned custom applications such as the order entry system for project "A" (we might call these "compiled" applications) would benefit from an object style programming model provided by Hibernate, or OpenJPA. But it may be that highly dynamic "metadata"-driven applications like the marketing analysis system developed by project "B" would require iBATIS or pureQuery (we might call these "interpreted" applications). Figure 4.6 graphically shows the relationship between these two project types and individual project requirements within the enterprise.

The possibility of choosing multiple frameworks implies that you should ensure that they can work together—adding the ability to integrate with other frameworks, such as JDBC, to your checklist of enterprise requirements to consider. Some requirements, like this one about integration, may be important for a number of project types —implying that a hierarchy of project types would be useful to minimize redundancy.

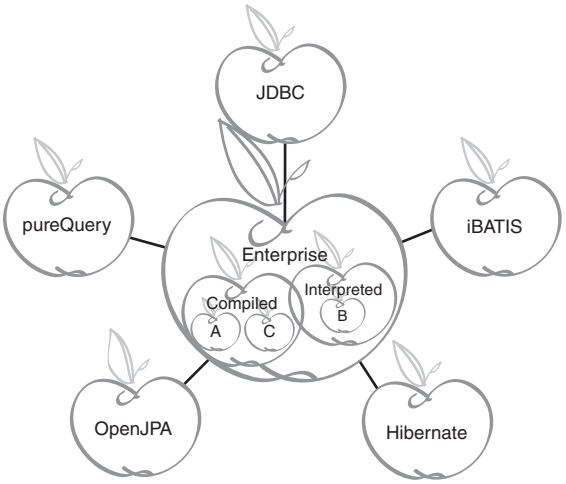



**Figure 4.6** Relationship of project types to the enterprise and project requirements.

## Ask Not If, but What and Why

But the problem with checklists of features phrased as "yes/no" questions is that for most interesting aspects of persistence (and life in general), the answers are not really black and white. Most of the time there are shades of gray.

For example, consider the question about whether a framework is available under open source. The description about Hibernate found in Wikipedia provides an answer:

> Hibernate is an object-relational mapping (ORM) solution for the Java language. It is free, open source software that is distributed under the GNU Lesser General Public License. It provides an easy to use framework for mapping an object-oriented domain model to a traditional relational database.

The answer was much more specific than just yes, and indicated that it was available under LGPL. We explore LGPL and other license agreements like GPL and Apache with respect to their impact on your ability to sell your applications in Chapter 2 "High-Level Requirements and Persistence." In short, there are usually trade-offs to consider for each one, and most frameworks would answer yes to only one in the set. So rather than asking a number of related "yes/no" questions, like

> *Is the framework distributed under GPL?*
>
> *Is the framework distributed under LGPL?*
>
> *Is the framework distributed under a license from Apache?*

it would be better to ask a single, more open-ended "what" question, such as the following:

> *Under what open-source licenses is the framework distributed?*

This single "what" question not only minimizes the number of questions you have to ask, but also makes it clear that you are evaluating an important aspect of your enterprise requirements.

## The Devil Is in the Details

This transformation of many related questions into one that is somewhat open-ended also illustrates that most often, it is not a question of "if" a given framework supports a requirement, but one of "how." In other words, the devil is in the details.

The underlying question we are after concerning the type of license agreements is this:

> *How is the framework distributed?*

A "how" question like this is usually too open-ended for an apples-to-apples comparison—leading us to substitute one or more "what" questions like that given previously. For another example, consider the third sentence from the Hibernate description again (bold italics ours):

> …it provides an ***easy to use*** framework for mapping an object-oriented domain model to a traditional relational database.

Certainly every framework would like to claim that it is easy to use, so asking a "yes/no" question is rather useless; a completely open-ended "how" question is nearly so:

> *How is the framework easy to use?*

One reason is that the notion of whether a framework is easy to use is a subjective evaluation that applies to the framework as a whole. A better "how" question is specific and objective:

> *How many steps does it take to achieve a given task?*

Within this context, the framework with the fewest steps across all the tasks evaluated would be considered the "best" with respect to ease of use. The questionnaire to drive the data gathering might start with a "what" question around a specific feature, coupled with a very specific question about how that particular feature is used for that task. For example, support for isolation levels could be objectively evaluated with the following two questions:

> *What isolation levels are supported?*
>
> *How are they specified?*

The details provided by the "how" questions make it easy to objectively compare the complexity of the solution with that of others—as long as the same example is used for each framework to minimize the open-endedness.

An evaluation based on this amount of detail allows you to apply Occam's Razor and choose according to its age-old best practice: *"All things being equal, the simplest solution tends to be the best one."*

## An Evaluation Template You Can Use

Now that you know how to conduct an objective evaluation and what general kinds of questions to ask, you have all the tools you need to create your own questionnaire and conduct your own evaluation. But because the devil is in the details, we go a step further and extract a questionnaire from enterprise requirements discussed in the previous chapters that you can use as a starting point for your own evaluations.

Our questionnaire is very comprehensive, with the following sections:

- Background
- Architectural Overview
- Programming Model
- ORM Features Supported
- Tuning Options
- Development Process for the Common Example

Each section begins with the major questions that the section is intended to answer, followed by specific subsections with more detailed questions.

### Background

This section is most closely related to Chapter 1, "A Brief History of Object-Relational Mapping," where we discuss the various frameworks that you might want to consider in your evaluation. Specifically, you want to be able to answer the following general questions:

*Why is the framework is popular enough that you chose it to evaluate?*

**Type of Framework**

*What approach to persistence does this mechanism support?*

**History**

*How did this framework evolve?*

### Architectural Overview

This section is derived from the discussion about business drivers in Chapter 2 and is intended to capture some of the more intangible "costs" of moving to that particular mechanism. A reviewer should be able to answer the following questions:

*What are the basic components of the framework and how they interact?*

*Show a high-level diagram that includes development tools and runtime servers.*

**Standards Adherence**

*What standards does this mechanism support?*

### Platforms Required

*What software and hardware platforms does this technology require for the runtime?*

### Other Dependencies

*What other Java technologies are needed in order to use this one?*

### Vendors and Licenses

*What vendors or open-source communities currently offer this framework?*

*For each one listed, what kind of license is available?*

*For each listed, how can you obtain a trial and/or production download?*

### Available Literature

*What kinds of educational material and reference guides are available? (Provide a link to various sources where available.) Consider using the following table format for clarity:*

| Title | Source | Description |
|---|---|---|
| | | |
| | | |

## Programming Model

The purpose of this section is to drill down to see how the framework can be used to support the basic usage patterns found in Java applications needing persistence discussed in Chapter 2.

*What is the programming model like in terms of components you have to develop and the language you use for each?*

*What is the "lifecycle" of a persistent object with respect to framework and application components?*

### Initialization

*What kind of initialization is required to access the framework API within your service implementation?*

*Show the code required to initialize the framework before a query of Customer with id = 100.*

*Show how to set up the framework with the Derby JDBC driver and also how to use an application DataSource with JNDI name jdbc/Derby/DataSource.*

### Connections

*What kind of connection model is supported?*

*Show an example of explicitly creating a connection with the same query used in the initialization section (if supported and not already shown in the initialization section).*

*Show how to set the database connection pool size to 20.*

### Transactions

*What kinds of transactional control are supported by the framework?*

*Show an example of grouping two update statements into a single user-defined transaction. If possible, use the following updates:*

1. Update a business customer with id = 100 such that businessPartner = false.
2. Update that customer's openOrder such that status = 'SUBMITTED'

*Show an example of configuring the framework to use global transactions with an application server (such as with JTA).*

*What isolation levels are supported?*

*For each that is supported, show the code required.*

### Create

*Describe the general approach to creating objects within the framework.*

*Show the code required to create a new ResidentialCustomer.*

*Show the code required to create a new Order owned by the Customer.*

### Retrieve

*Describe the general approach to queries and object retrieval within the framework.*

*Specifically, are dynamic queries supported?*

*Show the code required to retrieve a single Customer that matches a given primary key.*

*Show the code required to retrieve all Customers that are "active" (specifically, all CUSTOMERS that have an OPEN_ORDER).*

### Update

*Describe the general approach to updating objects within the framework.*

*Show the code required to change the quantity of a LineItem. Specifically, update a LineItem with a given customerId (assume it's already defined) to a quantity of 100.*

*Show the code required to change the OpenOrder attribute to Null.*

### Delete

*Describe the general approach to deleting objects within the framework.*

*Specifically, is cascaded delete supported?*

*Show the code required to delete a LineItem.*

*Show the code required to delete an Order and all associated LineItems (through a cascaded delete, if supported).*

### Stored Procedures

*Describe the general approach to invoking stored procedures within the framework.*

*Show the code required to call the following "swap order" Java stored procedure:*

```
SwapPojo orderIds = new SwapPojo();
orderIds.setFirst(orderId1);
orderIds.setSecond(orderId2);
dm.update("swap.order",orderIds);
```

### Batch Operations

*Describe the general approach to supporting batch operations within the framework.*

*What kinds of update operations are supported in batch mode?*

*For each operation supported, show an example.*

### Extending the Framework

*Describe any possible extension or plug-in points with brief examples of interfaces/APIs.*

*If the framework supports plugging into a distributed caching framework, you may want to defer this example to the distributed caching section.*

### Error Handling

*Describe the general approach to handling exceptions in the framework. If possible, consider including a class diagram showing the hierarchy in UML or your favorite notation.*

## ORM Features Supported

The questions in this section are derived from Chapter 3, "Designing Persistent Object Services," first focusing on the static definitional elements of the ORM problem. Specifically:

*How does the framework alleviate the object-relational impedance mismatch problem and make the life of the developer easier?*

*Are there any ORM features that are unique to this framework for common problems like object identity (such as key generation) or the load-time paradox (navigating large collection relationships)?*

### Objects

*Describe the general approach to mapping objects within the framework.*

*Show how you would define an Order object.*

### Inheritance

*What types of inheritance does the framework support?*

*For each type of inheritance supported, show how the framework is used to specify that a ResidentialCustomer inherits from an AbstractCustomer.*

### Keys

*What types of key attributes are supported?*

*Show how you would define that an Order has an auto-generated key.*

*Show how you would define that an Order has a simple integer key (if not already covered in the programming model section).*

*Show how you would define that a LineItem has a compound key consisting of an Order key and a Product key.*

*What key generation options are available? Specifically, does the framework offer key generation independently of the underlying database (for example, generating a UUID)?*

*What database-specific methods are supported?*

*Show how you would define that an Order has an auto-generated key.*

### Attributes

*What types of attributes are supported?*

*How are they mapped to the underlying database?*

*Show how you would define that a LineItem has a BigDecimal quantity attribute and how it can be mapped to the Derby type of* `BIGINT`

*Show how the* `Order.Status` *enumeration can be mapped to* `VARCHAR(9)` *of* `OPEN`, `SUBMITTED`, *and* `CLOSED`.

### Contained Objects

*What kinds of contained objects are supported?*

*Show how you would define that a Customer contains an Address object.*

### Relationships

*What "directionality" of relationships does the framework support?*

*What cardinality does the framework support?*

*Show how to use the framework to specify that an Order is related to a Customer.*

*Show how to use the framework to specify that a Customer is related to many Orders.*

*Show how to use the framework to specify that a Customer can reference at most one OpenOrder.*

### Constraints

*What types of constraints on attribute values does the framework support?*

*Can the framework leverage underlying database constraints?*

*Show how the framework is used to specify that a Product referred to by a LineItem must be in the Cataloged state. For this example, add a new field to Product called cataloged that is a Boolean.*

### Derived Attributes

*What types of computations are supported for "derived" attributes?*

*Show how you would define that a LineItem amount attribute is computed by multiplying the price by the quantity.*

*Does the framework support derived attributes that operate on properties from more than one (related) object?*

*If so, show how you would define that a LineItem amount attribute is computed by multiplying the Product price by the LineItem quantity.*

*Does the framework support derived attributes that represent operations across the elements of a collection?*

*If so, show how you would define that an Order total amount attribute is computed by summing the associated LineItem amounts.*

## Tuning Options

This section is derived from questions in Chapter 2 concerning quality of service IT requirements. You should be able to answer questions like the following:

*How does the framework make tuning at runtime easier without requiring code changes?*

*Are there any tuning features unique to this framework?*

### Query Optimizations

*What approaches to optimizing queries are supported that do not require changing the Java code?*

*For example, is it possible to change the number of rows returned by a query?*

### Caching

*What kind of caching strategies are enabled by the framework?*

*For example, does the framework have its own "single JVM" cache that can be configured?*

*If so, show how you would cache Products (because they change relatively infrequently).*

*How do you handle cache invalidation?*

*Does the framework support integration with a "third-party" distributed cache?*

*If so, show how that would be done within the framework.*

### Loading Related Objects

*What support does the framework provide for "hydrating" objects related to the target?*

*Does the framework support "lazy loading" such that specific related objects can be declared to be loaded only when explicitly referenced in the application code?*

*If so, show how you would configure the framework to indicate that the LineItems are lazy-loaded.*

*Does the framework support "preloading" such that specific related objects can be declared to be loaded at the same time that the referencing object is loaded (which usually results in a join operation on the database)?*

*If so, show how you would configure the framework to indicate that the Product related to a LineItem is preloaded whenever the LineItem is loaded.*

### Locking

*Describe the general approach to configuring locks on objects independently of the Java code.*

*What kinds of configurable locking strategies are supported?*

*Show how each would be enabled within the framework configuration options.*

*Does the framework support ordering of operations outside of the code to minimize deadlock conditions?*

*If so, show how this is done through configuration options.*

## Development Process for the Common Example

The purpose of this section is to show the end-to-end steps required to develop the services shown in the common example outlined in Chapter 3. The overarching questions include the following:

*Are there any development steps that are simplified by or unique to this framework?*

### Defining the Objects

*Describe any steps needed to map the Java objects to the underlying relational stores.*

*Specifically, are there any specific tools that need to be run?*

*If so, show when and how they get invoked.*

**Implementing the Services**

*Describe any special considerations for implementing services that are unique to this framework that may limit their portability.*

**Packaging the Components**

*Describe any special considerations for packaging the code components and the configuration components.*

*Are there any specific naming or project/directory structure conventions that need to be followed?*

*Are there any special tools that need to be run to compile or bind components?*

*If so, show when and how they get invoked.*

**Unit Testing**

*Describe any approaches unique to this framework for handling unit testing.*

**Deploying to Production**

*Describe any approaches unique to this framework for deploying the applications to production.*

*Specifically, are there features that make it easy to move from a system test to production environment with little or no configuration changes?*

*Are there any special tools that need to be run?*

*If so, show when and how they get invoked.*

## Making the Most out of Your Experience

Of course you will want to follow our earlier advice to keep it real and customize this questionnaire to the needs of your enterprise, taking into account your unique culture and requirements. For example, you may want to consider adding scenarios that are meaningful to your own enterprise as sections because they will give the best impression of the complexity of a given framework with respect to the functionality it delivers.

### Use the Questionnaire Early and Often

To help offset the cost of customizing the questionnaire, we recommend that you reuse it as a starting point for your project requirements. This approach will ensure that your more inexperienced project teams do not overlook a crucial requirement.

Also, constant use of the questionnaire on actual projects gives you lots of opportunity to validate them and keep them current. If a new project represents a new type or comes up with some new requirements, you should first add them to your enterprise questionnaire. Then consider whether to evaluate each project in progress with respect to the new requirements.

Depending on the situation, you may need to revisit your choice of persistence frameworks being used in your production applications—especially if you found a problem during development and testing that led to the new requirements to consider in the future.

We have seen a questionnaire like this one used in other ways, too. In one project, we used it to review the project team's choice of persistence mechanism. This questionnaire helped us to ask the right questions in a systematic fashion and get to the heart of the reasons for their choice.

### Record Your History So You Don't Repeat It

Whether you use this questionnaire in practice to conduct your own formal evaluations or review the work of others, make sure to record your answers and choices so that you don't forget them later and have to revisit.

A soft-copy version of the questionnaire in Microsoft Word is available on the download site, ibmpressbooks.com/title/9780131587564. You are welcome to use this template as-is or as a starting point for your own evaluation and review questionnaire.

We use this same template as a starting point for the next five chapters, which evaluate JDBC, iBATIS, Hibernate, OpenJPA, and pureQuery. We intend to have a Wiki section in the download site where questions can be added, and reviews of new frameworks (or older ones still in use) can be uploaded—so that we can all learn from each other and avoid repeating ourselves.

## Summary

Sometimes it is good to repeat yourself, especially in a summary. This chapter covered a number of best practices to follow when evaluating persistence frameworks for use in your projects. And if you can only remember "top ten lists" (with apologies to David Letterman), here are 10 key points to keep in mind:

1. *Be objective.* Labels like "better" or "best" have meaning only within a specific context of measurement and interpretation of the results.
2. *Be unbiased.* Measure against an independent standard rather than what a given framework chooses to advertise.
3. *Be practical.* With respect to providing an independent standard for evaluation, a union is good but your project requirements are better.
4. *Be comprehensive.* Develop a complete checklist of enterprise requirements as a union of those from previous projects.
5. *Be flexible.* Different project types have different enterprise requirements, so factor a separate checklist to choose the best framework for each project type.
6. *Be open-ended.* It is better to ask a single "what" question about an aspect of the requirements than many related "if" questions.

7. *Be specific.* Follow your "what" questions with a "how" question that clearly defines a scenario so that the details can be compared as well.
8. *Minimize redundancy.* Consider building a hierarchy of project types with common enterprise requirements checklists in the more "abstract" types.
9. *Maximize reuse.* Use your requirements checklists as a starting point for your new projects to minimize your analysis and design costs.
10. *Remain relevant.* Adjust the questions and reevaluate your choices based on actual project experiences.

This chapter also provided a detailed questionnaire that you can use as a starting point for driving and documenting your own formal evaluations and reviews. We use this questionnaire in the remaining chapters of this book.

## Links to developerWorks

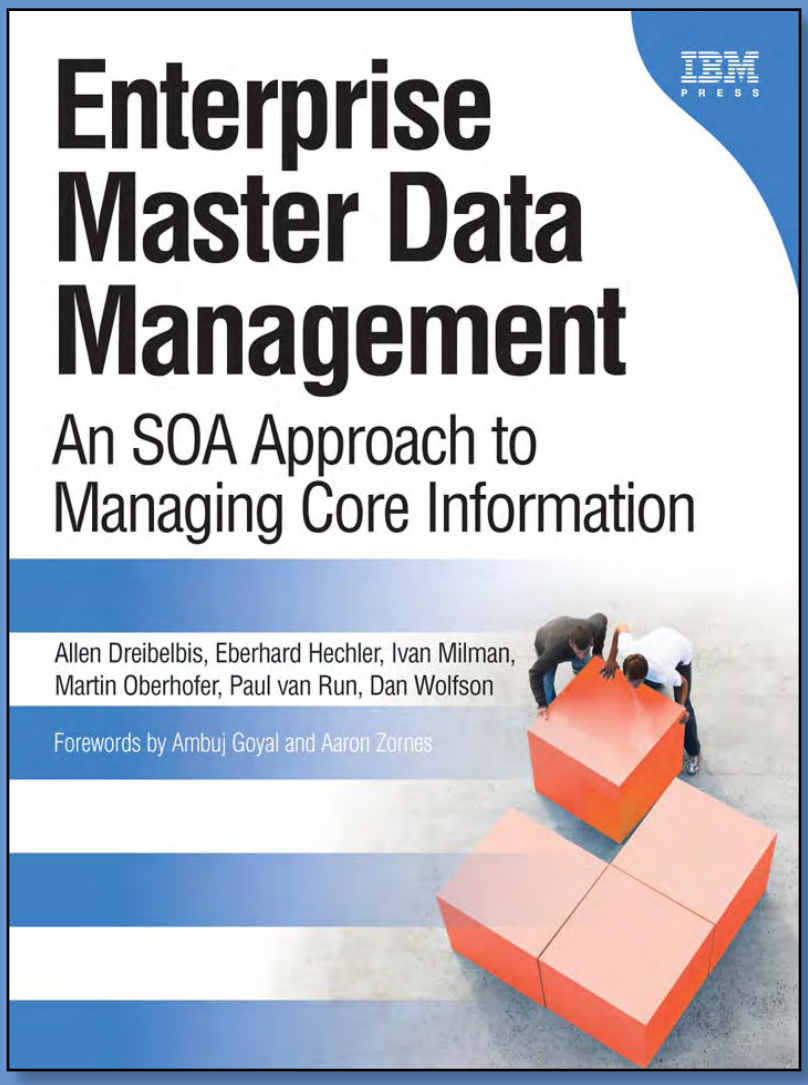
**A.4.1** *Developing Your Applications using SQLJ.*

An article by another ISSW consultant, Owen Cline, which describes practical uses of SQLJ in Java applications, including ones that use EJB components.

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0412cline/

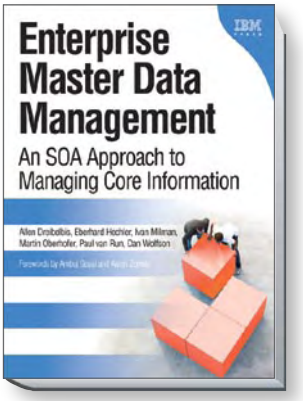
## References

[TopLink] *TopLink JPA*.
http://www.oracle.com/technology/products/ias/toplink/jpa/index.html



Allen Dreibelbis
Eberhard Hechler
Ivan Milman
Martin Oberhofer
Paul van Run
Dan Wolfson






# Enterprise Master Data Management

## An SOA Approach to Managing Core Information

©2008 ISBN: 0-13-236625-8

### Table of Contents



### About the Authors

**ALLEN DREIBELBIS**, Executive Architect in the IBM Software Group Worldwide Information Platform and Solutions Architecture Team, led the development of the MDM Reference Architecture.

**EBERHARD HECHLER**, Technical Enablement Architect for IBM Information Platform & Solutions, works with system integrators throughout Europe.

**IVAN MILMAN**, Senior Technical Staff Member at the IBM Software Group and IBM Master Inventor, focuses on security and governance in information management.

**MARTIN OBERHOFER**, Technical Consultant at the IBM Böblingen Lab, is a member of the Worldwide IBM Software Group Master Data Management Center of Excellence.

**PAUL VAN RUN,** IBM Senior Technical Staff Member, is responsible for architecting the MDM Server and WebSphere Product Center.

**DAN WOLFSON** is an IBM Distinguished Engineer and Chief Architect/CTO for the Information Platform and Solutions segment of the IBM Information Management Division of the IBM Software Group.

IBM Press™

# Chapter 2

# MDM as an SOA Enabler

In this chapter, we discuss the relationship between Master Data Management (MDM) and a Service-Oriented Architecture (SOA). We demonstrate how MDM and SOA complement each other to achieve the business and IT goals related to managing master data. Furthermore, we explain why we view MDM as an enabler for any SOA-style solution. This chapter is targeted at Enterprise Architects, Solution Architects, Lead Architects, and Chief Technology Officers.

> *You will waste your investment in SOA unless you have enterprise information that SOA can exploit . . . .*
>
> Gartner Group, 2005[1]

## 2.1 Overview

Service-Oriented Architectures are about supporting flexibility and reusability in the enterprise through careful design and implementation of technical and business architectures.[1] Much has been written on this popular subject—both at the technical and non-technical levels. In this chapter, we focus on the critical relationship between Service-Oriented Architectures (SOA) and MDM, and how MDM will enable SOA-style deployments. But why is this relationship critical? Well, the Gartner quote that appears before the first paragraph of the chapter gives us a starting point for our discussion. From a technical point of view, an SOA promotes the definition and reuse of modular services that invoke core functions that may be used in many business processes. One such function might be to return the address of a customer, given some identifying information. This sounds simple—but where

1. Simply put, a business architecture describes how the business is put together. It describes the value chains, business process, organizational structure, information systems, governance models, and so on that are present in the enterprise.

does my service get the customer information from? If we do not have an MDM Solution in place, we will likely have many different places where this information resides—and it is unlikely that they will always agree. So do we try to check each one and look for a consensus? Do we choose one and hope it is correct? Does it matter if the service is being called from an e-Commerce application or from the billing system? Does each of the customer information sources structure the data in the same way? And what about the user of the service—how does a service consumer know which services he or she can trust to return clean and current data?

If we had an MDM Solution in place—in this particular example, we are considering a Customer Data Integration (CDI) MDM Solution—then the answer is clear; and in fact the MDM System itself would likely provide the service definitions and implementations for us to use. In establishing an MDM implementation, we set the policies on which information is authoritative, how it should be used, and how it may be exposed. MDM therefore plays a pivotal role in an SOA strategy; it enables an SOA-based solution.

In short, the core issue here is that SOA can give a very process-based focus on developing component-based architectures and the usage of component-based modeling to map components to process tasks. Data is not a focal point here anymore because a service orientation through an SOA has taken its place. However, when data is redundant and inconsistent, you will get conflicting results based on which system you use to provide the SOA service or component. Data quality has in fact become the "missing child." MDM alleviates this problem by providing services that access high-quality, consistent and de-duplicated master data. It puts the data back in SOA.

In Chapter 1 we described the three principles of a successful MDM System. Such a system:

- Provides a consistent understanding and trust of master data entities
- Provides mechanisms for consistent use of master data across the organization
- Is designed to accommodate and manage change

Each of these principles has a role in the discussion of MDM and SOA. The first and second principles say that we can trust that the MDM System is providing authoritative information that can be used by other services and processes; in an SOA environment, we provide this *information as a service* (see Section 2.3). Beyond having trust that the MDM System provides consistent, accurate, and complete master information, the first and second principles also relate to the need for being altogether able to retrieve valid master data records enterprise-wide. "Consistent use" also refers to reusability of the data and its related services and the ability to deliver information in the right context, at the right time, to an authorized user, process, or application.

The third principle is also important to SOA environments. One of the key benefits of SOA is to provide business flexibility to better accommodate change. To support this, we need to provide technical flexibility at all levels—from the business processes that coordinate business functions to the services that implement these functions and the underlying information that these functions act upon. Therefore, to fully support an SOA environment, an MDM System should be designed to accommodate and manage change.

In a common ROI (Return on Investment)-driven implementation approach, MDM and SOA are applied in those spots in the enterprise where they provide the most business and technical flexibility for business components. The components can then be leveraged to improve business competitiveness and react quickly to change as well as reduce IT inflexibilities.

It is vital to notice that change is not limited to the IT domain only: Change is above all a business requirement that is addressed through the service orientation of the entire enterprise: A service-oriented enterprise (SOE). In other words, an SOE maps and translates into a more technically oriented Services-Oriented Architecture. The key converging technologies driving service orientation for the enterprise are the usage of XML, Web Services, Business Process Management, and SOA. An SOE requires a change in how IT interacts with the business and incorporates changes in organizational structures, business, and governance processes to better enable service orientation. An MDM System enables this service orientation both from a technical and a business perspective in order to deliver bottom-line business benefits.

As we have seen in the introductory Chapter 1, the need for Master Data Management is driven by concrete business needs, such as providing 360-degree insight into the customer base; building and operating a single source-of-truth; optimizing key business processes, such as New Product Introduction (NPI), which works best with a service orientation across enterprises.[2] Because of this natural affinity of MDM to the service orientation of an enterprise and even across enterprises, we should view MDM as a very convincing business motivation—perhaps even the first real business motivation—to actually implement a Service-Oriented Architecture.

MDM Systems should be designed to both support and to exploit Service-Oriented Architectures.[3] In this chapter we take a deeper look at this relationship. We start with an introduction to SOA followed by a broader look at some of the key ideas and principles behind SOA and their relationship to MDM Systems. We then take a deeper look at the notion of Information as a Service and how this concept extends the basic principles of SOA. Finally, we look at how MDM as a Service fits into a Service-Oriented Architecture.

Although this is not a book on SOA, we consider it necessary to describe some basic SOA concepts and characteristics. While this will undoubtedly be a somewhat incomplete overview from an SOA point of view, it will allow us to put SOA and MDM into perspective.

## 2.2 Brief Introduction to SOA

Although there are many good books available on SOA (e.g., [2]), we include here a brief introduction into the topic. A key tenet of Service-Oriented Architecture is to align the technical architectures and IT environment with the business goals of the organization.

2. In Section 6.3 of Chapter 6, we provide much more detail about NPI.

3. However, we have come across customers who want to implement MDM without SOA, mostly because they have not been able to make a good business case for the latter yet.

There are many different views of what exactly an SOA is, mainly because there are many different perspectives through which to view it. For a business person, SOA can be seen as an architecture supporting a business strategy geared at business integration, higher quality, cost reduction, and agility. SOA in this interpretation involves a collection of standardized, self-contained business services that are available throughout the enterprise or to external parties and that can be used individually or in a composed manner. SOA in this case is about modeling the business processes, gaining insight from these models, and using it to increase quality, agility, and resilience of the business. This process furthers the attainment of the business goals and gives the business the ability to change its processes in response to changing business circumstances. It also ensures that those changes are reflected more quickly into the business' information systems. This view is essentially centered on the service orientation of the business itself. And as we have pointed out earlier in this chapter, it may very well be that this business view of the service orientation—the SOE—does not necessarily result in any concrete adjustments or changes in the IT environment.

To a developer, SOA can be viewed as a programming model, which is by definition an abstract conceptual view of the structure and operation of a computing system. For a developer, the SOA programming model consists of standards, tools, and technologies such as Web Services. SOA in this perspective is more centered on the implementation of SOA components.

In this book we focus on technical architecture, and we therefore use the definition most appropriate for a technical Enterprise Architect. For such an architect, an architecture can be defined as the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution (as per IEEE Standard 1471-2000). Using this definition, we can define an SOA as follows:

> *SOA is an integration architecture in which components are available through services. These services are available through platform-neutral interfaces and communication protocols. They encapsulate application functionality to be delivered to the service consumer. They are loosely coupled and based on a formal definition or contract.*

In order to be useful and valuable for the enterprise, business services need to have a number of important characteristics:

- They need to be performed in a repeatable and sustainable manner. Repeatability and sustainability allow for the prolonged existence of the enterprise. A business cannot survive if its main business services cannot be repeatedly performed in the same manner.
- They need to adhere to the business' rules, guidelines, and principles in order to fit within the business' plans and goals. They also need to adhere to external rules, regulations, and laws.
- They can be decomposed in finer-grained business services or composed into business processes.
- They need to be managed, measured, and monitored to operate them, to ensure they perform within the norms, and to gauge their effectiveness.

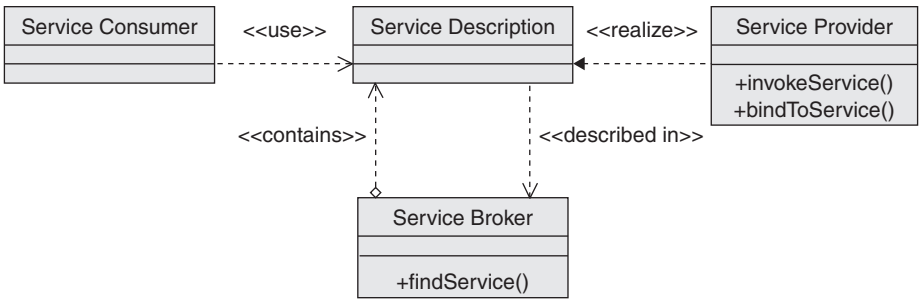



**Figure 2.1** Conceptual Model of an SOA Architecture.

- They need to be agile, or adaptable, to allow the business to change and adapt to changing circumstances and markets, product innovations, or changing regulations.
- They need to be cost-efficient to ensure the business processes they are used for can be managed cost-effectively.
- They need to allow for integration both within the organization and with outside parties. These days, very few companies can operate as isolated entities and few even have isolated processes; an increasing number of tasks, or even complete processes, are outsourced.
- They need to provide a high level of quality in many ways, such as their availability, their speed of delivery, and the quality of their deliverables.

We can describe a conceptual model of an SOA architecture using four basic concepts. There is the **Service Provider** who realizes the service by providing an implementation for it, and who publishes a **Service Description**.[4] The **Service Consumer** can either use the Service Description directly or optionally find it using the Service Broker. It can then bind to and invoke the service from the Service Provider. The **Service Broker** component (also known as a Service Registry and Repository) is optional in this model. This component is a core repository and system of record for service definitions and policies. It is responsible for making (Web) service interfaces and implementation access information as well as metadata around policy enforcement available to potential requestors. Figure 2.1 is a depiction of the coherence of these four basic concepts.

### 2.2.1 SOA Enterprise Architecture

Before we discuss the relationship of SOA and MDM, let's try to understand the key characteristics of an SOA enterprise architecture. Figure 2.2 is a high-level depiction of a layered approach.

4. We use "service description," "service definition," and "service contract" interchangeably.

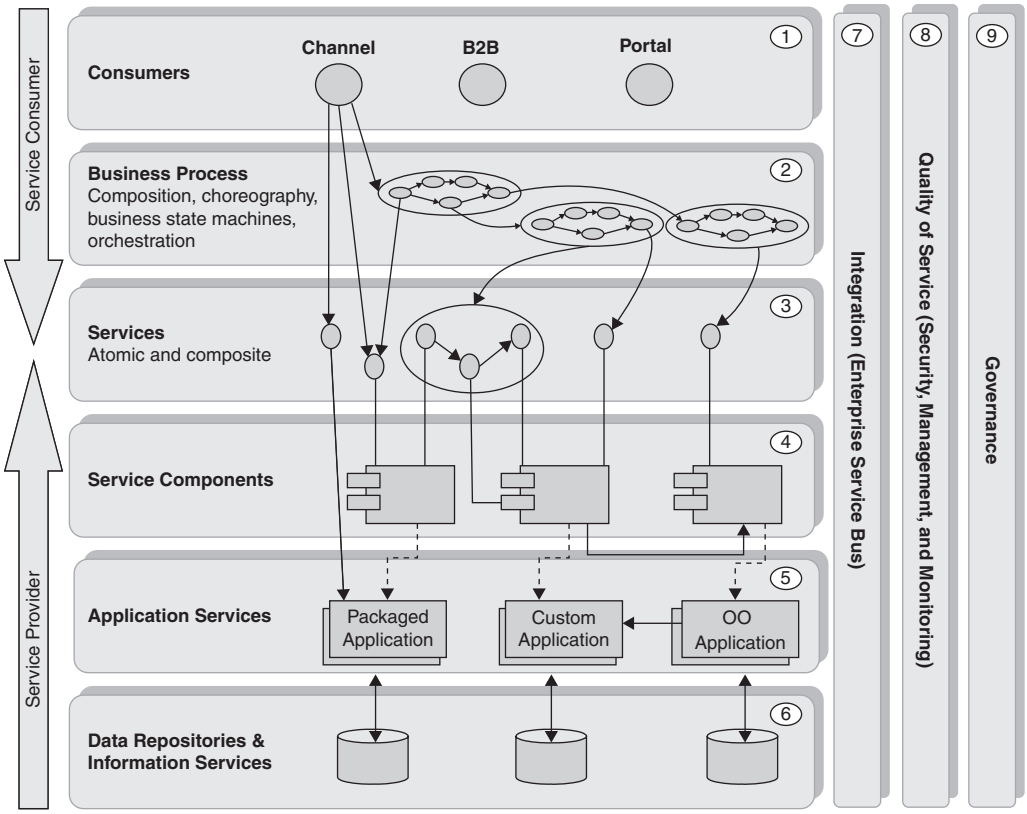



**Figure 2.2** SOA Enterprise Architecture (the figure is an adaptation of similar figures from [4] and [5]).

We can describe the layers of an SOA Enterprise Architecture (for more details see [4] and [5]) as follows (the numbers in the descriptions are referenced in Figure 2.2):

> **Layer 1—Consumers:** This layer consists of User Interfaces (UI) and other end consumers of the services. These consumers can be part of many different channels, such as call centers, services kiosks, Web self-service, data stewardship functions, and business-to-business (B2B) integrations. SOA decouples the user interface from the service components, and this layer is often considered out of scope for SOA discussions. However, the mapping of user tasks in the UI to services in the SOA is an important component of the end-to-end SOA solution. Different users with dissimilar roles need access to diverse tasks and services. The presentation of similar data from identical services might need to be different based on the user's role and the user's device. We are also seeing a convergence of standards in this area, such as

Web Services for Remote Portlets (WSRP)[5] and other technologies, that seek to leverage Web Services at the application interface or presentation level.

Placement of Logic:

- No business logic should be present in the user interface layer. The UI should use the services available from the business process layer only.
- Should be limited to presentation only. No direct access to the data; access is available only through the services.

**Layer 2—Business Process (Composition, Choreography,[6] Business State Machine,[7] and Orchestration[8]):** Business processes in general, and specifically the service orientation of business processes, is key to building an enterprise SOA. The compositions and choreographies of services exposed in layer 3 are defined in this layer. Services are bundled into a flow through orchestration or choreography and thus act together as a single application. These applications support specific use cases and business processes. This layer allows for adequate composition and orchestration of services in the context of a given set of business scenarios with corresponding use cases. Here, visual flow composition tools can be used for the design of application flow.

Placement of Logic:

- Should be limited to workflow design and implementation that is defined by the business analyst.
- Business process should have no visibility or understanding of the underlying operational systems.

**Layer 3—Services (atomic and composite):** The services that the business chooses to fund and expose reside in this layer. They can be discovered, for example, by querying a registry, or be statically bound and then invoked. Services can be atomic, in which case they are self-contained and they do not invoke services themselves, or they can be aggregated into a composite service. This service exposure layer also provides for the mechanism to take enterprise-scale components, business unit-specific components, and in some cases project-specific components and externalize

5. Web Services for Remote Portlets (WSRP) is an OASIS-approved network protocol standard designed for communications with remote portlets.

6. Choreography describes externally observable interactions between services, peer-to-peer collaborations, by defining their observable behavior. Web Services Choreography Description Language (WS-CDL) is a language for describing multiparty contracts.

7. Business State Machine is an event-driven business application in which external operations trigger changes that guide the state machine from one discrete mode to another. Each mode is an individual state, and this mode determines what activities and operations can occur.

8. Orchestration relates to the execution of specific business processes. WS-BPEL is an OASIS standard and a language for defining processes that can be executed on an orchestration engine.

a subset of their interfaces in the form of service descriptions. Thus, the enterprise components provide service realization at runtime using the functionality provided by their interfaces. The interfaces get exported out as service descriptions in this layer, where they are exposed for use. They can exist in isolation or as a composite service.

Placement of Logic:

- Should be limited to discovery and allocation of the service request to the appropriate service component(s).
- Mostly technical rules, such as appending access control tokens, logging, error handling, and making routing and binding decisions.

**Layer 4—Service Components:** This is the layer of enterprise components that are responsible for realizing functionality and maintaining the Quality of Service (QoS) of the exposed services. These special components are a managed, governed set of enterprise assets that are funded at the enterprise or the business unit level. As enterprise-scale assets, they are responsible for ensuring conformance to Service Level Agreements (SLA) through the application of architectural best practices. This layer typically uses container-based technologies such as application servers to implement the components, workload management, high availability, and load balancing.

Placement of Logic:

- Predominantly, the rules needed to map a service request to a sequence of call(s) to operational systems.
- Includes the technical mapping, binding rules, and dependency rules for combining underlying operational capabilities into understandable service interfaces.

**Layer 5—Application Services:** This layer consists of legacy systems (existing custom-built applications), existing CRM, ERP, and other packaged applications and existing object-oriented system implementations. The composite layered architecture of an SOA can leverage existing systems and integrate them using service-oriented integration techniques. We can see the importance of legacy integration in Chapter 3 (on MDM Reference Architecture), in Chapter 5 (covering MDM architecture patterns), and specifically in Chapters 6, 7, and 8 (covering MDM Solution blueprints and MDM integration blueprints). Any MDM Solution needs to be able to integrate with legacy systems and needs to ensure master information synchronization between the MDM System and those legacy systems that contain master data of a chosen domain. In short, application services as part of an SOA enterprise architecture ensure the necessary integration with the MDM System.

Placement of Logic:

- Includes all of the business and technical rules already encapsulated into the existing application.
- Behavior in this layer is constrained by the application context boundaries.

**Layer 6—Data Repositories & Information Services:** This layer consists of the structured (analytical data, operational data, master data, metadata, etc.) and unstructured (documents, images, etc.) enterprise data managed by various data providers such as databases, data warehouses, directories, or file systems. This data is available directly by accessing the underlying data providers, but it can also be trapped in operational systems and therefore may only be available through their interfaces or UIs. The information services consist of data services, metadata services, content services, master data and information integration services, and analysis and discovery services. We discuss the information services that pertain to MDM in detail in Chapter 3, MDM Reference Architecture.

Placement of Logic:

- Includes all of the information services associated with Database Management, Content Management, Information Integration, Master Data Management, Analytics, Metadata Management, and Data Governance.
- Provides the Information-as-a-Service (IaaS) functionality (more on IaaS in Section 2.3).

**Layer 7—Integration (Enterprise Service Bus):** This layer enables the integration of services through the introduction of a reliable set of capabilities, such as intelligent routing, protocol mediation, and other transformation mechanisms, which is often described as the Enterprise Service Bus (ESB). On the other hand, an ESB provides a location-independent mechanism for integration.

Behavior:

- Replaces the many-to-many connectivity problems with a bus architecture. Every application connects to the bus. Changes in protocols now only impact one connection point.
- Handles the inter-application communication, that is, which messages need to go to which destinations.

**Layer 8—Quality of Service (Security, Management, and Monitoring):** This layer provides the capabilities required to provide a certain QoS, such as security, performance, and availability. This consists of integrated and background processes through sense-and-respond mechanisms and tools that monitor the health of SOA applications, including the standards implementations of WS-Management[9] and other relevant protocols and standards that implement QoS for an SOA.

Placement of Logic:

- Monitors the behavior of all of the other layers in the SOA to ensure QoS.
- Performs the monitor task in the SOA lifecycle (more on this task in Chapter 9).

9. WS-Management is a Web Services management standard defined by the Distributed Management Task Force (DMTF).

**Layer 9—Governance:** This layer represents the people, processes, and procedures required to maintain trust and control of the information services we establish. Details are provided on governance in Section 2.3.3 and Chapter 9, Master Data Management and Data Governance.

## 2.2.2 SOA Characteristics and Master Data Management

If we examine the definition of architecture again, we will recall that it is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution. There are many excellent books on the topic of Service-Oriented Architectures, and we refer you to them if you want to dive deeper into specific SOA design principles, architectural aspects, or implementation methodologies. For details on SOA business value, planning, and the enterprise roadmap, see reference [2]. In the next section, we focus on a number of characteristics of SOA services and discuss their influence in an MDM environment.

- Service reuse
- Service granularity
- Service modularity and loose coupling
- Service composability
- Service componentization and encapsulation
- Compliance with standards (both common and industry-specific)
- Services identification and categorization
- Provisioning and delivery
- Monitoring and tracking

### 2.2.2.1 Reuse

Reuse refers to the use of existing software assets in building new software. Code reuse has been a software engineering goal from the early days of programming. Its target is to reduce the development time of a system, increase the reliability of the code by using code that's already been tested, which increases the maintainability of the code, and in general reduce the overall cost of developing software. In its earlier forms, reusable subroutines and procedures found their way into programming languages, and eventually entire reusable software libraries were created. With the advent of Object-Oriented Design and Programming (OOD & OOP), the focus shifted to creating reusable classes. Object-oriented principles of inheritance, encapsulation, abstraction, composition, and polymorphism all tried to make code more reusable. While there have been successes, especially in the areas of framework class libraries such as J2EE and .NET, reuse of business software at the class or object level remains very difficult, and initiatives in that area, such as the IBM San Francisco project, or Microsoft's Business Framework, have not been very successful.

Why is that? In order to create truly reusable business objects, there has to be an agreement on a single definition of an entity for an enterprise. In the case of a generic object such as a string or a hash table, there is very little additional context required to come to such a definition, but for many of the main business entities, such as customer, product, order, or account, there is much more context required, which makes defining entities a much harder task. The exact

definition of an order to one business unit in an enterprise might be considerably different from that of another business unit. The definition of a customer for the commercial business side of an enterprise, where it refers to a company, is very different than the one for the private business side, where it refers to a person. When such a definition is used across multiple organizations within the same industry, or even across industries, this problem is magnified, and it is therefore much more complex to create such objects for general reuse.

In addition, when these components need to be integrated with those from other vendors, or those built in-house, the attached contexts need to be synchronized further, which complicates the integration. And even if agreement is reached, business change is inevitable, and business object definitions that hold true one day might be invalid the next day. Rapidly responding to business changes by adapting the involved information systems is a recurring IT challenge. But changing existing business object definitions used in many systems requires an enormous amount of rework. Lastly, even if such agreement can be reached, it is often difficult to convince the developers of an organization to actually make use of the available reusable business objects, and any enforcement of company requirements to use them takes considerable effort as well.

The element of reuse for an SOA architecture is the service itself. Service reuse is a big driver in SOA adoption, and survey results from, for example, the analyst firm Hurwitz & Associates [6] show that almost 90% of IT executives point to it as their number one priority here. When we map service reuse as an SOA guiding principle to a business requirement for a business task or service, we are mostly talking about IT cost reduction, as was the case with object reuse. However, in the case of services, reuse also leads to greater consistency. The more users use a service, the more consistently business processes within the enterprise can be performed.

As we saw earlier, rather than providing objects for developers to reuse, an SOA exposes reusable services that a developer can access. The internal structures are not exposed, and developers deal only with the service definition, the service contract. The underlying implementation can therefore be changed without affecting any of the consumers as long as the contract is preserved. The true value of SOA enablement becomes much more apparent after the services are being reused in many different business scenarios and when they do not need to be redesigned, rebuilt, or retested. Reuse of existing code assets should also lead to an increase in code quality. Moreover, the repeatability of the business processes is increased as well; the business processes can become more consistent through the reuse of services. This is something that is more unique to services reuse and not a direct result of object reuse. Reusing the entire business process instead of just the business objects allows for an entire new level of abstraction: that of business process modeling.

Reuse in SOA can be subdivided into two separate concerns. First, the reuse of existing IT infrastructure, such as legacy mainframe COBOL[10] applications and custom or packaged applications through their SOA enablement. The second is the reuse of the SOA services themselves.

10. COBOL (COmmon Business-Oriented Language) is one of the oldest and most widely used high-level programming languages still in active use, initially created in 1959. COBOL is targeted at business, finance, and administrative systems.

Reuse of existing legacy operational systems is often mandated. In many cases, it can be prohibitively expensive to attempt to replace all of the existing applications with newly created ones that were built according to an SOA. Therefore, existing applications may need to be SOA-enabled in order to integrate them into the SOA Enterprise Architecture. Many of the IT assets required to enable an on-demand[11] business already exist and have been supporting the business for years or even decades. Enabling these IT assets to participate in integrated business processes is key to improving business responsiveness. Enabling existing IT assets requires discovery, analysis, and enablement phases: Which applications are already available in the enterprise, which ones need to participate in the SOA ecosystem, and how can they technically be SOA-enabled? Several asset analyzer tools are available in the industry to help you find and identify COBOL copybooks,[12] CICS® regions,[13] and transactions, database assets, and message middleware queues and calls. After identifying such systems, an impact analysis of the effect of SOA enablement should be performed. Obviously, adding the load from an online Web system to a legacy application through an SOA enablement can have significant impact on the current performance of such a system. The actual SOA enablement of existing application is highly dependent on their technical architecture. It could, for example, exist in a proxy or wrapper routine that acts as a services interface between a COBOL application and an SOA infrastructure. Again, there are many products available to help with this phase, which is another indication of the importance of this type of reuse.

Let's now look at the second type of SOA reuse, which is reuse of the actual SOA services themselves, after they have been created. The business context that is required for the achievement of successful business object reuse is also a problem for business services reuse. Gartner Group mentions that only 20% of developed services are likely to be reused [7]. Yet some things do change when we compare service reuse within an SOA environment to business object reuse in a purely object-oriented environment. An SOA offers a developer reusable services as opposed to a reusable object class. For example, instead of a common customer business object, there are exposed services that allow for the creation, search of, inquiry into, maintenance of, and deletion of customer data. This eliminates some of the problems around reuse. There is less of a need to come to an agreement on a single definition of a business object. Instead, the services client needs to conform to the definition of the business entities that the services expose. If it is different from the one the client requires, then it must map its own view into the one exposed by the services. Because the services are the only way by which access can be gained to the underlying

11. On Demand Business: "An enterprise whose business processes—integrated end-to-end across the company and with key partners, suppliers and customers—can respond with speed to any customer demand, market opportunity or external threat." Sam Palmisano, CEO, IBM.

12. A Copybook is analogous to an "include file." It's a section of code that is copied into several different programs. It is most often used to define the layout of a physical data section, either for file input/output or as a work or communication area.

13. CICS (Customer Information Control System) is a transaction server that runs primarily on IBM mainframe systems. A CICS Region is a named collection of resources controlled by CICS as a unit. Such resources can include programs, queues, files, transactions, and so on.

data, enforcement of the reuse of existing services is easier. And as long as the existing externally visible services interfaces (the "contract") remains unchanged, the underlying implementation and business rules can change completely, which makes the entire architecture more agile.

The main driver for reuse remains, as it was in the object reuse case, a firm commitment to a service quality strategy for the enterprise. With top management support, strong commitment, and diligent effort, several companies are reporting significant savings from SOA-based reuse [8].

David Chappel mentions the following roadblocks to service reuse [7]:

- **Commitment:** As with object reuse, service reuse is hard to achieve without a corporate commitment to quality. Top management support, strong commitment, and diligent effort are required to achieve significant savings from SOA-based reuse.
- **Predicting the future:** How can one accurately predict which services are required in the future? Targeting existing needs is much less risky.
- **Variation in requirements:** There are often many variations in the exact definitions of what the different service consumers require. Some might need additional pieces of information, and some might not be allowed to see certain information.
- **Ownership:** There is not always a lot of incentive for the creator of a service to share it with others, both from a service as well as from an information asset point of view.

#### 2.2.2.2 Master Data Management and Reuse

Master Data Management focuses on providing SOA services[14] for a clearly delineated set of objects in the enterprise: the master data. By limiting the scope in this manner, the effect of some of the roadblocks to reuse just discussed can be diminished, as follows:

- **Commitment:** MDM enablement is often positioned as an enterprise strategy. The whole premise of MDM is to elevate master data entities to a higher level within the corporation, away from the bounds of being locked into their proprietary, stovepipe,[15] applications. The business targets of an MDM implementation are often about obtaining a better view of an MDM entity, such as a customer or a product, across the organization. By elevating the adoption of an MDM implementation to this level, it is necessary to get a higher level of management commitment, and therefore usage of MDM services can be more easily enforced. The decision to pursue an MDM strategy is often more of a business decision than an IT decision. This fact brings with it more commitment from the business to promote reuse and

14. In practice, MDM can be implemented without services, but for the discussion on reuse, it is the SOA aspect we are focusing on.

15. A stovepipe application is typically a legacy information system in which the components (code, data, etc.) are so tightly bound together that they can no longer be upgraded or refactored. The application must be maintained until it can be replaced in its entirety. The application typically serves only one particular purpose or line of business.

service quality. With MDM, SOA enablement does not just represent a technology and architecture advancement—it also enables a whole new array of master data usage possibilities (more on executive sponsorship is presented in Chapter 9, Master Data Management and Data Governance).

- **Predicting the future:** When considering building an in-house MDM System from scratch, the future usage of the system is rather hard to foretell. In this respect, it suffers from the same hindrance to SOA reuse. Commercial MDM Systems available in the market differ in the way they provide SOA services. Some merely provide tooling to build MDM structures and services; others provide large sets of pre-built SOA services "out of the box." In the latter case, these services are built based on the requirements and experiences gathered from many different client implementations. This somewhat reduces the uncertainties around future use of the MDM System and the associated services. In addition, most commercial MDM Systems come with tooling to augment existing services within their frameworks, thus allowing for an easier adaptation to new business requirements. The critical factor in this process is the ability to augment a service without breaking the service contract with current service consumers, especially because any MDM Solution is to be viewed in the context of an evolutionary approach. MDM implementations often start with a single master data domain (e.g., only Product or only Customer), a limited set of services, and overall a limited scope of master data usage within the enterprise. Over time, many MDM implementations grow beyond their initial boundaries, and therefore it is important to be aware of potential changes early on and to proactively anticipate growth and adjustments over time. This plays an even bigger role in MDM than in traditional SOA implementations. To add to this complexity, multiple versions of MDM services may exist at one time, because requirements evolve at a dissimilar pace for different client systems. Using a strong commercial MDM product allows an enterprise to have some confidence that its initial MDM implementation can mature over time into a more comprehensive system.
- **Variation in requirements:** Even in an MDM implementation, different service consumers will have varying requirements around the MDM services they require. One of the biggest challenges is arriving at shared definitions for business objects, processes, and entities being enabled using MDM. Commercial MDM Systems typically offer frameworks and tooling to provide for additions and extensions of services and underlying entities, which makes the variation in requirements from the consumers more manageable. Alternatively, MDM services are often built with configurable behaviors, for example, a single inquiry service that can be invoked with various inquiry levels to return slightly different sets of data. This variation in requirements is also related to the master data domain: It could very well be that the initial scope of the MDM implementation is limited to a chosen master data domain that has to be widened to embrace additional master data domains in order to address the needs of other service consumers. Similar variations in requirements are related to the MDM methods of use and the MDM deployment or implementation style. In Chapter 5, MDM Architecture Patterns, we come back to these variations of requirements as they relate to MDM methods of use and MDM hub patterns.

- **Ownership:** In an MDM implementation, the services are often owned by an enterprise-wide IT organization. It governs the usage of these services, thereby reducing contentious issues around their ownership between different departments or business units. The issues now shift back to a data ownership question: Who owns a piece of data and who has access to it? Most commercial MDM Systems provide data visibility (security) and entitlement (ownership) functionality for their services, entities, and data. This enables different user communities to use the same services in order to access different data elements. All MDM deployment styles will consolidate the management of master data and naturally result in a stronger centralization of master data services. More on this topic is presented in Chapter 9, on Master Data Management and Data Governance.

MDM further enables the reuse of SOA by defining, delivering, and enforcing the quality of the most important enterprise entities: the master data. It is done by following the concepts of "Information on Demand," which are to deliver consistent, accurate information in the right context at the right time to an authorized user or application. By encapsulating data quality functionality in the services, we can ensure that any access to the data has to follow the same data quality rules. Incorporating metadata knowledge into this equation can further enhance the data quality notion. Metadata about the MDM service can tell us where the data came from, what its latency is, and how trustworthy it is. Higher quality guarantees lead to higher confidence in the data and the services, and this confidence, in turn, will be an important incentive to reuse the services.

#### 2.2.2.3 Service Granularity

Granularity refers to relative size of the components that make up architecture. It relates to the level of detail that is considered in a particular piece of code. Finer granularity offers greater potential for parallelism and hence improved speed, but it also leads to greater overhead of synchronization and communication. In SOA, granularity refers to the granularity of the business services. Fine-grained services provide a small amount of business functionality, such as some basic data access, while coarse-grained services are constructed from finer-grained ones to meet a more complex business need. This construction typically includes program logic and business logic to manage the service composition. This pattern of decomposing coarse-grained services into finer-grained ones is sometimes referred to as a fractal pattern; the pattern keeps repeating itself at any level of detail.

There are several key areas to consider when determining the level of granularity of a service: Business Mapping, Performance, Transaction Scope, and Management and Governance. Let's discuss them in detail.

- **Business Mapping:** The SOA service granularity should resemble the business service granularity as described in the overall business design. The granularity should allow the SOA service to fulfill an entire business task and nothing more. Additions should only be made when absolutely necessary. This approach promotes reuse of the service because it matches the business thinking.
- **Performance:** Finer-grained services typically execute faster and contain less information, while coarse-grained services are the opposite; they generally take longer

than and deal with more information. Because we are dealing with remote invocations, network infrastructures, and overhead, in most SOA implementations this is an important performance consideration. If the services are too fine-grained, too much time is lost in network round trips, but if the granularity is too large, it could harm concurrency in the system. Careful consideration must be given to this tradeoff to determine the performance optimum.

- **Transaction Scope:** Services should be self-contained and should not require service state to be maintained in between calls. If the services are too fine-grained and multiple calls are required to complete one business task, there is a potential for inconsistencies in the event of system failures that occur before all of the calls are completed. If the granularity is too coarse, substantial rework might be required if one of its finer-grained services fails. Service calls should never perform more than one transaction—for this same reason.
- **Management and Governance:** A proliferation of fine-grained services can lead to increased costs of management and governance in addition to failing to achieve the significant economies of scale associated with reuse of coarse-grained services that provide more business value. We consider it important to harmonize fine-grained services with business requirements.

#### 2.2.2.4 Master Data Management and Service Granularity

Let's now try to map the preceding discussion on service granularity to our topic of Master Data Management in an attempt to understand how MDM relates to the four key areas of Business Mapping, Performance, Transaction Scope, and Management and Governance. MDM influences the granularity of the services in various ways, and it therefore has an effect on these key areas of consideration.

- **Business Mapping:** MDM Systems typically offer both fine-grained (singular) and coarse-grained (composite) services. Using the Customer Data Integration (CDI) MDM area as an example, a fine-grained service might be "addEmailAddress," which only adds an e-mail address to an existing customer record, while a coarse-grained one might be "addCustomerWithMultipleContracts," which adds a customer with multiple names, addresses, identifications, contracts, and so on. It is up to the service consumer to choose which service best fits its requirements. The service consumers can decide to go with one of the predefined services or decide to build their own. If needed, new services can be custom-composed out of existing fine- and coarse-grained ones, or existing services can be extended to best service the consumers' needs. Providing coarse-grained services to the service consumer reduces the efforts required for composition. Coarse-grained MDM services contain all of the inter-service business logic and are optimized and pre-tested.

  Using another example from the Product Information Management (PIM) MDM area, a fine-grained service might be "addProductAtribute," which adds another attribute to an already existing product, while a coarse-grained service might be "categorizeProducts," which links a set of products to a product category in the MDM System.

As you can imagine, there are also fine-grained and coarse-grained services that address the linkage of the different master data domains. For example, a cross-domain, coarse-grained service that encompasses both CDI and PIM functionality might inquire about a client and the products he or she purchased. Determining the right balance between fine- and coarse-grained business services is the key to establishing a strong and adequate linkage to the business domain.

- **Performance:** Because of their centralized position within the enterprise, MDM Systems often need to be able to scale to very high usage and data volumes. When several applications use an MDM System to obtain their master data, a badly performing MDM application can quickly become a critical bottleneck for many enterprise processes. To ensure performance, both fine- and coarse-grained services need to be extensively tested and tuned to achieve nonfunctional performance requirements. Coarse-grained services tend to put more strain on concurrency and therefore require special attention during their design. Insert and updates, especially coarse-grained ones, normally perform slower than inquiries, but inquiries often constitute a larger percentage of the transaction mix. The transaction mix used for testing has a very significant impact on its outcome. Because service usage patterns vary widely between implementations, commercial MDM Solutions can require implementation-specific performance optimizations such as custom indexes or pluggable search queries. A key to delivering optimum performance is the use of the right balance of fine- and coarse-grained services in a deployment and the appropriate customization of the MDM Solution. That is, any vendor providing MDM Systems with a set of predefined fine and coarse-grained services needs to rely on the customization and the deployment, usually done by a System Integrator. A System Integrator will understand the customer requirements and will translate these needs in terms of transaction throughput, data volume, end-user response time, growth considerations, and other MDM Solution aspects, which will then result in a customized set of fine- and coarse-grained business services to be used for that particular implementation.
- **Transaction Scope:** In a stateless MDM System, no transactional information needs to be maintained between service calls. This significantly improves scalability of the system. Clustering of application server nodes as a means to achieve higher scalability is much less complicated and performs better without the need to maintain transactional state across the cluster. As we have seen earlier, when we discussed the transaction scope from a more general SOA point of view, the right balance of fine- and coarse-grained services is important to ensure completion of a required business task in a single service call. The transaction scope of an MDM service needs to encompass all of the data changes that are part of that operation. In many cases, however, the MDM service invocation can be part of a larger business process, and the transaction control function might not lie with the MDM System itself. An enterprise business process might span service calls across several systems, one of which could be the MDM System. In this case, the MDM service needs to participate in the larger transaction scope. If the entire business process completes successfully, all of its services, including the MDM ones, should commit their changes,

and if something goes wrong, they should all roll them back. An MDM System needs to allow for both situations, one where it is in charge of the transaction scope and one where it is but a participant.

- **Management and Governance:** MDM services often have to be available in their fine-grained form. These are usually services to maintain a relatively small part of an MDM entity, for example, updating an address for a customer. One reason for this centers around integration, for example, other systems have to be able to use fine-grained MDM services as part of their own business processes. In cases where an entire business flow, such as a "New Business Process (NBP)" or "New Product Introduction (NPI)," can be handled by an MDM System, the granularity of the services is more coarse-grained. As a consequence, an MDM System tends to have a fairly large set of services, both coarse- and fine-grained. Different consumers of the services tend to use different subsets of these coarse- and fine-grained services. While an initial MDM implementation might use only a limited set of services, this number tends to grow over time when more processes start to use the MDM data through the MDM services. This may also be influenced by adding additional master data domains and widening the scope to include additional MDM methods of use. Having all of the services related to MDM centralized in one location does allow for many advantages with regard to management and governance. It is easier to enforce a standardized set of granularity levels per master data entity and an associated naming standard. An example would be where every entity has the same set of "get," "update," "delete," and "search" services defined on it. Services such as security, data visibility and entitlements, auditing, and data quality can be managed centrally. Proliferation of services can be further reduced by, for example, offering inquiry levels for all of the inquiry services (the same inquiry service can return more or less information based on this predefined setting). In many cases, control of the MDM System is in the hands of a central enterprise architecture group, which helps to keep tighter control over the services that are being developed.

#### 2.2.2.5 Modularity and Loose Coupling

Modularity is the property of a computer system that measures the extent to which that system is composed out of separate parts, called modules. Modularity in systems development allows for separation of concerns, the ability to do modular development, and other advantages. A system is considered more loosely coupled if the relationships between two random modules in that system occur through well-defined interfaces. A system is less loosely coupled if there are more direct relationships between modules of the system. A module has an interface describing the elements it provides and requires. This interface is used to hide the underlying implementation. Loose coupling is one of the key areas for SOA, because it guarantees the reusability of the services. Loose coupling in an SOA means that the client of the service is independent of the implementation of the service. The client does not need to know the details of the service implementation (which platform, which programming language) in order to communicate with it. And as long as the

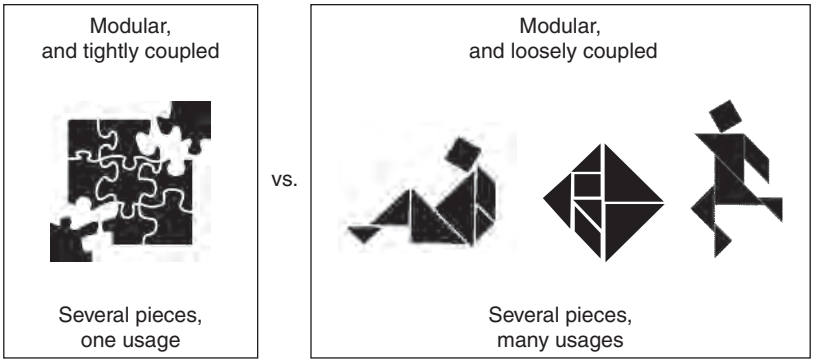



**Figure 2.3** Tight coupling vs. loose coupling.

interface of the service remains the same, the consumer is shielded from any implementation changes.

Let's use a simple example to illustrate the earlier statements: A traditional analogy for modularity is that of a jigsaw puzzle [9] where modules are like the pieces of the puzzle; when they are fit together in the correct way, they offer a solution to a problem. A jigsaw puzzle is an example of tight coupling, where all of the modules fit together in only one predetermined way and the solution is a single, static result. This reflects the usage of modularity in traditional applications, where multiple modules delivered a single business process. A related analogy for a loosely coupled architecture like SOA would be a Tangram puzzle, where the puzzle pieces can be fit together in many different ways to construct a variety of end results. Figure 2.3 shows tight coupling vs. loose coupling.

This Tangram analogy is illustrative of the composition of loosely coupled SOA services. Loosely coupled, well-designed SOA services can be composed into multiple different business processes, in diverse ways. This brings many advantages, especially when combined with more effective reuse. New applications can be constructed more quickly by using existing modular SOA services. It also introduces additional complexities, because in the design, construction, and testing phases, multiple usages of the service need to be considered.

It is important to note that loose coupling can happen at a number of different levels. The commonly used example is that of consumers using service definitions and thereby only loosely coupling themselves to the service implementations.[16] It's important to note, though, that the same implementation can be used by multiple different service definitions and that a single service definition can be tied to multiple implementations. See Figure 2.4 for an example: Service A and Service B both provide services defined on implementation B, and Service A has two possible implementations, either A or B.

16. You can also read "component" where "implementation" is used here.

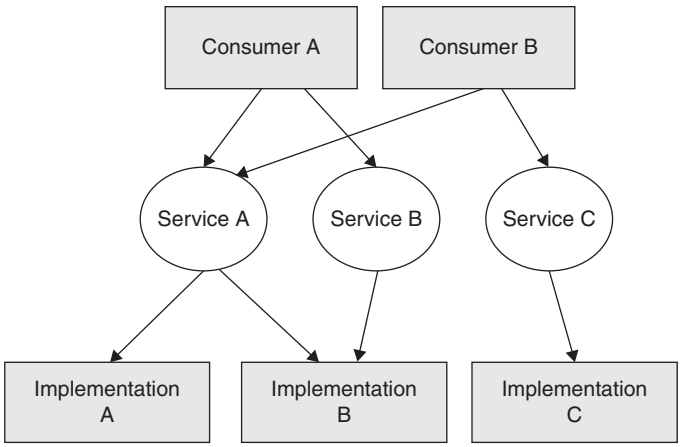



**Figure 2.4** Loose coupling in a Service-Oriented Architecture.

### 2.2.2.6 Master Data Management Using Modularity and Loose Coupling

MDM Systems benefits from the usage of modularity and loose coupling in various ways.

- **Internal modularity and loose coupling within the MDM application:** As with any large system, a well-designed MDM System benefits from modularity in its own design. In the traditional sense, modularity improves the maintainability of the system, it helps manage its complexity, and it offers separation of concerns and other benefits in the development lifecycle. However, there is another specific advantage of modularity and loose coupling for an MDM System and that is its adaptability. By constructing various components of an MDM System in a modular way, the system can adapt more easily to changing requirements. By configuring a particular module differently or by replacing it with a different one, the system can be changed without impacting the base functionality. Practical examples here are the use of a business rules engine and associated business rules to modify the behavior of the MDM services, or the transparent use of an external data standardization tool by the MDM services to improve the data quality. Other examples of such pluggable and configurable components are data standardization rules, message adapters, matching engines, and scripting tools.
- **External modularity and loose coupling of the MDM Services:** By definition, SOA services are modularized, loosely coupled elements themselves, but there is a deeper level of modularity that is of interest here. In a well-designed MDM System, coarse-grained SOA services are compositions of modularized fine-grained ones. The fine-grained services in turn are reused in various coarse-grained ones because they are loosely coupled. This offers not only the advantages of code reuse but the advantages of reuse of the business logic defined at the finer-grained level. For example, a coarse-grained business service to add a customer might use a finer-grained service to add an e-mail address under the covers. Because of the internal modularity and reuse of the *addEmail* service, any business rules defined at that level are

automatically applied at the composite coarse-grained, *addCustomer*, level. So if the fine-grained *addEmail* service is enhanced with data quality functionality to verify the correctness of the e-mail address, any composite coarse-grained services that use *addEmail* obtain the same improvement.

- **Modularity of the total MDM Solution:** An MDM System is often embedded into a larger business solution; it is not a solution on its own. It is therefore extremely important to be able to take the MDM puzzle piece (Tangram, not jigsaw) and fit it with all kinds of other puzzle pieces to construct your overall business solution. Many applications today have their own version of master data, and you want to decouple the consumer side from the provide side so you can exchange the provider without impacting the consumer and vice versa. This is important when you want to move from a legacy environment to a strategic architecture that leverages master data.

Modularity and loose coupling is tightly connected (pun intended) to the concept of composability, which we discuss next.

#### 2.2.2.7 Composability

Composability is a system design principle that deals with assembling multiple self-contained components into new combinations. A system is highly composable if components can be selected and assembled in various combinations that yield new valid systems to satisfy specific user requirements. For an SOA system, this means that the SOA services need to be composable into higher-level services. This process is referred to as Service Orchestration and Choreography (see layer 2 of the SOA Enterprise Architecture in Section 2.2.1). The composite services are assembled as process services as defined in the business process layer of the logical SOA architecture. This is typically done by using a combination of workflow and state machine composition approaches using a language like Business Process Execution Language (BPEL).

The essential attributes that make a component, such as an SOA service, composable are:

- **Containment:** The component can be deployed independently, and while it can cooperate with other components, it is not dependent on those, and they can be replaced by others.
- **Statelessness:** Each request to the component is treated as an isolated transaction; there are no dependencies on earlier transactions.

Services need to be self-contained; within a single request, they need to fulfill all of the required steps to complete a business function. They should not be dependent on previous or subsequent service requests. For example, it would be considered bad design to create separate services to commit or roll back previous services. In addition, SOA services should not maintain state between requests. Maintaining state leads to the creation of interdependent services, lack of containment, and reduced isolation of concerns. Statelessness and self-containment go hand in hand; if a service does not maintain state, it is not capable of transferring such state to another service. It has to complete the business function on its own. Composability requires the service owner to publish an unambiguous contract for the service

with clearly defined inputs and outputs. Collections of services can then be coordinated and assembled to form composite services.

#### 2.2.2.8 Master Data Management and Composability

MDM does not differ much from a typical SOA application in this area. Good MDM design requires the same attention to self-containment and statelessness of the services to increase composability. In many cases, self-containment also relates to the fact that an MDM service should be able to be executed as a single unit of work. All of the database tasks a service performs either are all committed, or they are all rolled back. So the execution of the service has a predictable result (success or failure) and is not dependent on other previous, concurrent, or subsequent services. If this service is used in a composition, the transaction scope often has to change to the scope of the encompassing service. The success of the database updates of the finer-grained service is now dependent on the success of the composite service. Even in this case, it is recommended to ensure that the logic of the internal, finer-grained service is self contained. An obvious anti-pattern in this case would be the existence of specific commit or rollback services.

Statelessness of the services is also beneficial to an SOA-based system in order to handle larger volumes of service requests. Because there is no need for the transfer of state between subsequent calls, the application can be scaled up by providing more application server nodes that serve up the same service. Scalability that approaches a near linear progression is desired, to ensure that a system can scale efficiently. There are other aspects that influence scalability, like network bandwidth or hardware scalability, but the absence of service state is an important contributor.

It is important to note that the statelessness of the services is different from the statefulness of the MDM application. Of course, the MDM System itself needs to maintain state in order to store and retrieve master data. This state is basically maintained in the database, but there are other stateful application components as well, such as registries, application caches, and property files. When designing these components for an MDM System, we also have to be mindful of the consequences of maintaining state, which pose a different set of issues.

#### 2.2.2.9 Componentization

Any large system as a whole consists of different parts or components. That is, looking at a system from a lower level yields a more detailed view and will naturally allow recognizing certain parts and components that the system is made of. The same is true for an SOA enterprise architecture: We are talking about modularization and gaining an understanding of the key parts of the architecture. The individual components will consist of a set of functions that have a certain affinity to each other. It's almost like grouping similar functions or services and "isolating" them in defined components, where these components will interact with each other. This is very much related to the earlier section on composability. Componentization has to do with the isolation of functions and design concerns. And in doing so, we are enabling composability.

#### 2.2.2.10 Master Data Management and Componentization

In mapping componentization to MDM, we need to identify the key components of an MDM System. Later on, in Chapter 3, where we describe the MDM Reference Architecture

(as well as in the blueprints presented in Chapters 6, 7, and 8), we dive into the architectural and solution aspects of MDM Systems. We will discuss then, on a more detailed level, the components of an MDM System. For the sake of the present discussion—trying to understand MDM in the context of SOA—we see specific architectural components. This is, of course, a simplification. However, looking at MDM at a high-level, we consistently recognize these components:

- **MDM services:** These are the fine- and coarse-grained services, which we discussed earlier. In essence, these are services to maintain master data as the "single source of truth," and services to support different MDM methods of use (operational, collaborative, and analytical).
- **Master Data Repository:** Depending on the MDM hub pattern (see Chapter 3 for details), this Master Data Repository is either fully materialized or contains a minimum of master data. In any case, this repository contains the instance master data from chosen master data domains (e.g., customer data), or it assembles the knowledge regarding how to assemble instance master data from different sources.
- **Information Integration Services:** This component encompasses important services used to build the MDM hub and to operate the MDM System. For instance, to build the MDM hub requires services to extract relevant data from legacy systems, to cleanse and transform the data, and to load the data into the MDM hub. Operating the MDM System requires ongoing information integration services such as information synchronization services.
- **Line of Business (LOB) Systems:** In any MDM Solution, we find a number of existing LOB systems, which the MDM System needs to work with. The boundaries of this collaboration could very well go beyond the enterprise to include collaboration with systems from other enterprises. Zooming into this collaborative infrastructure, we could even identify further subcomponents. These components play a strong role in the collaborative MDM method-of-use.
- **Third-Party Data Service Providers:** These components play a strong role in MDM Solutions. There will often be a number of external service providers that will be used to enrich master data from different domains. As an example, take a CDI-MDM hub of any bank, where customer data will be improved or validated through external data service providers, for instance, through credit rating companies such as Dun & Bradstreet (D&B).
- **Process Manager:** In the case of MDM Systems, there are many industry-agnostic, and industry-specific, business processes that require a well thought through orchestration of tasks that integrate usage of other components. MDM Systems consist of components that encompass these process models for further customization and optimization.
- **Connectivity and Interoperability Layer:** In general, there are many different ways to provide for connectivity and interoperability. In the case of SOA, this layer consists of an enterprise service bus, which provides different styles of integration. And specifically with regard to MDM, this layer provides the infrastructure that allows for the process- and application-focused information integration that is so essential for any MDM Solution.

You may very well have noticed that—in talking about MDM—we consider MDM services and Information Integration services as two important components of an MDM System, but there are other components as well. In Chapter 1, we discussed the various aspects of multiform MDM, and later, in Chapter 3, we elaborate on MDM hub patterns, which are very much related to MDM implementation styles. An interesting feature of MDM componentization is that these components can be consistently identified regardless of the various multiform MDM themes. In other words, regardless of the master data domain, the MDM method-of-use, and the MDM implementation style, these components are required to make up an MDM Solution. We also should like to point out that componentization in the MDM space is essential, because MDM is usually implemented enterprise-wide, meaning that it addresses a chosen master data domain in an enterprise-wide fashion with a need for collaboration and information synchronization with numerous other legacy systems. This can only be done based on a thorough understanding of key components of MDM Systems. As was stated previously, componentization goes hand in hand with composability. And in the MDM case, componentization provides a specific value regarding the evolutionary aspect of MDM Solutions. That is, componentization enables reuse of key components that are required to integrate other master data domains, or to implement additional MDM methods of use. It thus enables growth efficiently and with the right Total Cost of Ownership (TCO) in mind.

#### 2.2.2.11 Compliance with Standards

Compliance with industry-accepted open computing standards to support interoperability is an essential aspect of the service orientation of an enterprise and of an SOA architecture. Proprietary system interfaces and APIs, programming models and access methods, communication protocols, and so forth, don't really have a big play here. As we mentioned earlier, much has been written on SOA; we therefore just mention a few of the key standards that are important in the SOA context. Some of the important standards are *Extensible Markup Language* (XML); *Web Service Description Language* (WSDL), an XML-based language for describing Web Services; *Simple Object Access Protocol* (SOAP), a network protocol to exchange data between systems; the *Java Message Service* (JMS) API messaging standard; Web Services specifications, such as WS-Security, WS-Addressing, and WS-Policy; the *Business Process Execution Language* (BPEL), an XML-based language to describe business processes. These standards are driven by numerous consortia (e.g., OASIS) and standards development organizations (e.g., ISO).

#### 2.2.2.12 Master Data Management and Compliance with Standards

When we look at MDM from the perspective of whether there is compliance with technology standards, there isn't really any significant differentiating factor: Assuming that the MDM Solution is based on an SOA architecture, the earlier consideration on open technology standards applies as well. However, there is more to MDM, specifically in an industry context. Because of MDM's integrated, centralized position within an enterprise, an MDM System needs to be able to conform or adapt to industry-specific standards that are present. Of the MDM Systems available in the market, some are specialized for specific industry verticals, which allows them to target specific industry standards. Others are more generally

applicable and need to come with means to adapt them to the standards present in a particular industry vertical. In the retail industry, for instance, the *Global Data Synchronization* (GDS)[17] standard plays an important role. This is a standard that is part of the GS1 Global Data Synchronization Network (GDSN) set of standards, which is driven by the GS1 organization. In the insurance industry, the standards driven by ACORD play an important role. ACORD is a nonprofit association whose mission is to facilitate the development and use of standards for the insurance, reinsurance, and related financial services industries. Specifically, in the exchange of master information across enterprises (insurance companies, brokers, agents, distributors, and so forth), ACORD-based eForms management and exchange also has significant importance.

As we have stated previously, for any MDM Solution, electronic integration with business partners occurs in a very industry-specific manner that is based on common standards. SWIFT, EDI,[18] and RFID are just a few additional examples from the financial services sector, retail, and other industries. HIPAA and HL7 are common standards for healthcare.

#### 2.2.2.13 Services Identification and Categorization

Services Identification is part of the service-oriented modeling and architecture process. It consists of a combination of top-down, bottom-up, and middle-out techniques of domain decomposition, existing asset analysis, and goal-service modeling to determine the required services in a particular enterprise situation [5]. After identifying the services, it is important to **classify or categorize** them into a service hierarchy. Classification is needed to determine the layering and composition of services, and to determine the interdependencies. A typical enterprise might have over 100 coarse-grained sets of services that in turn could be composites of finer-grained, lower level services that in turn could depend on particular data services, and so on. Obviously, such a huge set of available services requires classification to be available for reuse by business modelers or IT. Several different classification views, or taxonomies, of the services can exist in parallel to give different points of view. Commonly used ones classify services—by business category, process, area or function, level of complexity, or business entity. Composition and classification help alleviate the performance, maintenance, and governance problems associated with a proliferation of too many fine-grained SOA services. Coarse-grained services tend to provide more business value to the enterprise and allow for economies of scale[19] to be achieved. However, some number of fine-grained services are still required to provide for special consumer requirements such as creating custom composites.

---

17. In Section 6.3 in Chapter 6, we describe the *Global Data Synchronization Solution* Blueprint for Retail, and we describe GS1, GDSN, and GDS in much more detail.

18. EDI is used both to refer to a set of standards to structure electronic data exchange and to refer to the implementation and operation of systems and processes for creating, transmitting, and receiving EDI documents.

19. Economies of scale characterize a situation where an increase in scale leads to a decrease in the per unit cost. For coarse-grained services, their increased complexity is outweighed by their increased value.

#### 2.2.2.14 Master Data Management and Services Identification and Categorization

MDM Services can typically be categorized by their usage type (collaborative, operational, or analytical) and their domain (customer, product, account). Within these categorizations, there can be additional subcategorizations of the services based on their business functionality (e.g., de-duplication, task management, lifecycle management, hierarchies). For example, in the case of an operational CDI Hub with Customer as its main domain, we can have services for demographic information, data stewardship, financial profiles, relationships, identifications, locations, contracts, and so on. In a multiform MDM Solution that supports multiple domains, there would typically also be cross-domain services that can be categorized in more than one domain category. The final categorization criteria is the composition level of the service: Is it a simple, single-task service or a richer composite with additional business logic and value that performs a more complicated business task?

#### 2.2.2.15 Provisioning and Delivery

A key aspect of an SOA is provisioning and delivery, which has many different aspects. One of them is provisioning and delivery of the deployment environment in terms of providing the infrastructure for the SOA Enterprise Architect to actually deploy the anticipated solution. This deployment has to be executed according to defined Key Performance Indicators (KPI), administrative and operational policies and procedures, performance characteristics, and many other metrics. Another key aspect is the delivery of a service according to a set of specifications, such as a given Service Level Agreement (SLA). The provisioning and delivery of such a service requires a process to determine required resources, such as hardware, network, and memory.

In general, provisioning and delivery in the context of an SOA architecture ought to be viewed as a specific capability that is to be delivered under the umbrella of the management of resources, as one of the following three SOA management layers:

- **Business Service Management**, which provides for service-level planning, business-impact monitoring, and prioritization of event management.
- **Composite Application Management**,[20] which provides support for securing information about the SOA environment, flow content analysis, end-user response time monitoring for service requests, service problem diagnosis, and application trace that you can then pass back to your development environment.
- **Resource Management**, which enables orchestration, provisioning, infrastructure health monitoring, and event automation.

20. Composite Application, although not limited to the scope of SOA, in this context it is a term used to refer to an application built by combining existing services. Its functionality comes from different sources within the SOA, such as individual Web services or legacy systems with Web services wrappers.

#### 2.2.2.16 Master Data Management and Provisioning and Delivery

What does this mean in the specific context of MDM? Let's examine the provisioning and delivery of the deployment environment, and of the actual services.

- **Provisioning and delivery of the deployment environment:** Analogous to general SOA-style solutions, delivering MDM Solutions requires a rather comprehensive deployment environment. MDM deployments come in all sorts and shapes, similar to what we would observe with typical SOA deployments. But even the largest businesses typically deliver and deploy MDM Solutions in a very phased, multiyear, multiproject approach. They typically start with a particular domain, MDM style, line of business, or geography and grow the solution out over time to a much wider scope. "Big Bang" MDM implementations are rare. Common contributing factors here are priorities, risk mitigation, availability of resources (funds, labor), and internal political factors and mandates within the enterprise. Such a phased approach requires a combination of a strong strategic vision and a series of directed tactical steps. The main focus needs to remain in order to achieve the ambitious and distinguishing characteristics of MDM Solutions—for instance, to maintain a single enterprise-wide "source-of-truth" for a master data domain. To be successful in this journey, the road traveled there needs to lead through a series of incremental steps, each delivering a ROI on its own, in order to maintain the project's momentum.
- **Provisioning and delivery of the services:** Provisioning and delivery of MDM business services, similar to general business services in an SOA, are to be performed in order to allow information provisioning in the context of specific business needs. In a typical first phase MDM deployment, only a subset of all of the available MDM services will be required: The scope of the project is limited and therefore so is the set of required services. This phased approach of MDM service availability brings certain maintenance and governance requirements. For instance, service implementations potentially need to be augmented in future deployment phases to serve additional needs from new consumers, but at the same time, current consumers of the services should not be impacted by future changes. Backward compatibility of the augmented services is a critical factor here. Commercial MDM Systems typically have provisions to accommodate for this need. Despite the typically phased approach, MDM is still an enterprise-wide concept, and it will therefore also require an enterprise provisioning and deployment strategy. Over time, many of the enterprise's systems will have to use MDM services to perform part of their business functions, or at least use them to inform the MDM System of certain changes. The services infrastructure of the MDM System has to be able to scale to these future demands.

#### 2.2.2.17 Monitoring and Tracking

Comprehensive monitoring and tracking of the entire service landscape is vital for any SOA-style solution. Distributed systems increase the risk of failure by eliminating siloed systems and replacing them with sets of services available across a network. Such systems require comprehensive system-wide monitoring and tracking as well as operational visibility to guard against failures and downtime. Monitoring and tracking includes but is not limited to areas such as business impact monitoring; infrastructure health monitoring; monitoring

performance of service request execution, including end-user response-time monitoring; SLA monitoring; database monitoring; monitoring the entire IT service environment; and so forth. Because we don't really intend to write a book on SOA, we do not dive into this any further. But the interested reader is of course encouraged to study the referenced literature for further details on this topic. The main point here is that SOA systems can consist of many components and be very complex. Such systems require extensive monitoring and tracking, and we now discuss if MDM adds any other requirements here.

#### 2.2.2.18 Master Data Management and Monitoring and Tracking

In order to highlight some MDM-specific monitoring and tracking aspects, let's structure our discussion according to the following multiform MDM characteristics. At first glance, this discussion may sound a bit artificial. However, you will see that these characteristics will nicely facilitate the discussion and will highlight the distinctiveness of MDM when compared to SOA. These topics are not exclusive, meaning that a combination of the following monitoring and tracking characteristics may have to be considered:

- **Master data domain:** Always considering that the focus on a specific master data domain is to build and *maintain* a consistent, enterprise-wide single version-of-the-truth, monitoring and tracking—beyond what should be done for any conventional SOA solution—needs to encompass this requirement of maintaining consistency, completeness, accurateness, and currency of master data. This therefore requires a means to establish a baseline on data quality (e.g., using profiling), an ability to periodically track for improvements or degradations, and a means to act if problems are detected. This should all be governed by a set of data quality policies and processes.
- **MDM methods of use:** Just to repeat: We have operational, collaborative, and analytical MDM methods of use. Each of them have unique monitoring and tracking requirements when compared to conventional SOA solutions:
  - The operational method-of-use requires monitoring and tracking of all master data business services, far beyond performance, end-user response time, and other more conventional metrics. Monitoring of operational MDM business services needs to include the why, when, where, and how of changing master data. The reason for that is the strong linkage of the MDM System to the legacy infrastructure and the need for master data synchronization.
  - The collaborative method-of-use extends monitoring and tracking beyond enterprise boundaries and needs to factor in other systems and third-party service providers. It also brings additional monitoring and tracking requirements around the workflows required for the collaborative processes as well as the import and export requirements of collaboratively authored data.
  - The analytical method-of-use, which goes beyond the traditional BI and data warehousing schema to address, for example, KPI measurements, cross- and up-sell opportunities, and threat and fraud-related scenarios, needs to comprise use case-specific and even industry-specific monitoring capabilities. An example is the need to monitor the business impact and timeliness of identified and

returned records, individuals, or transactions that are linked to threat and fraud scenarios. Monitoring in analytical MDM needs to be responsive to changing business needs (e.g., additional threat scenarios).

- **MDM deployment styles:** Depending on the chosen deployment style (and related MDM hub pattern[21]), changes to master data can occur at different systems, and the MDM hub itself is only one of these systems; changes can also occur at any of the many existing legacy systems. Thus, monitoring and tracking need to be implemented according to these different styles. In essence, this is a specific flavor of MDM, which calls for a unique implementation of monitoring and tracking capabilities. An example: The MDM business service "addEmailAddress" could be submitted against the MDM hub, or against one or several of the legacy systems. Monitoring and tracking need to be sensitive about the variations of MDM business service execution that are linked to the various MDM deployment styles.

## 2.3 Information as a Service

### 2.3.1 Information as a Service, Introduction

Information is the key commodity upon which an enterprise runs. If the information stops flowing, the enterprise stops. If the information takes too long to flow from one part of an organization to another, then the business suffers. Information is stored, shared, analyzed, transformed, and consumed. Therefore, just as we seek to describe, implement, and consume discrete business services that may be reused in an SOA environment, we also need to consider how to describe, implement, and consume Information as a Service (IaaS). Without Information as a Service, processes are tightly coupled to the data. Such tightly coupled processes are brittle, and they do not adapt well to continuously changing business requirements. They also lead to:

- **Inconsistencies in the view and the packaging of the data:** Because each process creates its own custom access to data, the view of the data across those processes is inconsistent. For example, one process might get its customer data from a different place than another. The way the data is packaged typically also differs from process to process, which complicates integration efforts.
- **Inconsistencies in the rules applied to the data:** Because the rules are captured in stovepipe applications, they are tightly connected to the data, and they are not shared among processes. For example, data validation rules or the calculations involving that data can vary from process to process. This reduces the quality of the data.
- **Multiple points of maintenance:** There is duplication of logic across the applications that access the data. This creates multiple points of maintenance for the same logic. This is complex, error-prone, time-consuming, and expensive. For example, each process needs to use the same logic to standardize telephone numbers or verify a product ID.

21. For details on these MDM hub patterns, see Chapter 5.

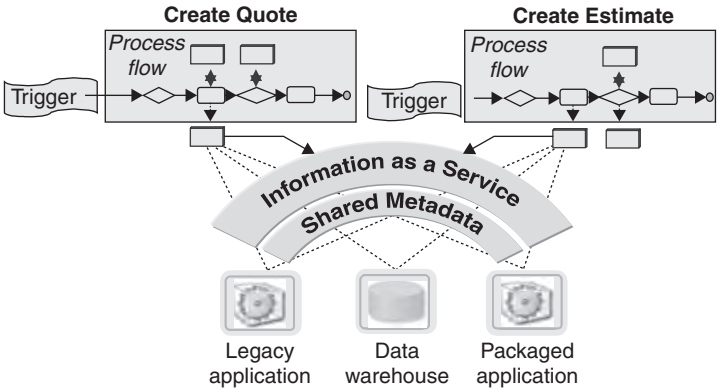



**Figure 2.5** Information as a Service.

Without a foundation of usable information, your service-oriented architecture is just a loose confederation of abstract business processes. It's your business information that delivers the value to your SOA. Information as a service is about providing a new level of services that helps add value to information contained in data sources across an organization. By treating information as a service, organizations can improve the relevance and cost-effectiveness of their information, making information available to people, processes, and applications across the business and improving the operational impact that information can have in driving innovation.

### 2.3.2 Information as a Service, Concept

Figure 2.5 illustrates the concept of Information as a Service (IaaS). In this example, the CreateQuote and CreateEstimate process flows at the top both need access to data stored in a legacy application, a data warehouse, and a packaged application. Instead of building services that access these data sources directly, as shown by the dotted lines, Information as a Service provides an intermediate layer that abstracts the process flows from the manner in which the data is stored. The foundation for IaaS is the shared metadata repository.

Information Services provide facilities to create this layer of information virtualization. They can roughly be categorized into Analysis and Discovery Services,[22] Master Data Management Services, Information Integration Services,[23] Content Services, Data Services, and Metadata Services. We will not describe all of those in detail here, but we will focus on MDM in the context of IaaS in the next section, Section 2.4.

22. For example, query, search, reporting, mining, operational intelligence, embedded analytics, metrics.

23. For example, Extract Transform Load (ETL), Enterprise Application Integration (EAI), Enterprise Information Integration (EII).

### 2.3.3 Information as a Service, Characteristics

Using Information as a Service helps clients identify, access, manage, secure, and deliver information in real time regardless of the type of information or the platform on which it is stored. It ensures consistent packaging of data, consistent application of rules to the data, and centralized control and maintenance. To treat information as a service, one needs to agree on a number of characteristics of the information [10]:

- **Definition:** The structure and the semantics (the meaning) of the information needs to be well defined and commonly available. The heterogeneity of the information is transparent; the consumer does not need to understand the diversity of the data sources or formats. Understanding the structure and semantics of information exchanged by a service is the first step in being able to safely and effectively use the information service. We need to understand not only the format of the messages exchanged between the requestor of the service and the provider of the service, but also what the different elements of the message really mean, which is part of the metadata of the service. In an SOA environment, the structural information is typically described by a combination of the Web Services Definition Language (WSDL) and XML Schema Definition (XSD). However, to properly interpret the information being conveyed by a service, we often need more information—to know the relationship this entity has with others, or to know the domain of values that an attribute is allowed. In short, we need authoritative documentation that describes what the information is and how it should be used. This documentation could be written or could be managed by a metadata tool. These characteristics are not unique—they are important to most SOA services—but it is especially important to information services because the focus is on the proper use and understanding of the information.
- **Quality:** The quality of the information needs to be known, and the integrity of the data needs to be ensured for retrieval as well as update scenarios. The source of the information needs to be known, and so does its currency. This "information about the information" is what constitutes part of the metadata. After we understand the information service and what it means, we next need to establish the quality of the information to be served. Information quality is a very broad topic with an extensive literature. We recommend a couple of very good books (see [11] and [12]). Data quality aspects were discussed at length in Chapter 1.
- **Governance:** Changes to the service and the underlying information need to be governed in a uniform and consistent manner. Data Governance addresses the issue of how to maintain trust and control of the information services we establish. Through a combination of business processes, people, and technology, we need to define and implement policies that control access to information and information services. Authorized users need to be able to audit the execution of information services and to manage and coordinate changes to the information services and structures. Chapter 9, Master Data Management and Data Governance, is devoted to the topic of data governance.

## 2.4 MDM as a Service

Now that we have introduced the information as a service concept, in Section 2.3, we now apply this concept to the master data domain. MDM is a key component of the concept of Information as a Service. Just like IaaS, you can interpret MDM in this sense as an additional tier in a typical three-tiered architecture in which you have a presentation tier, a business logic tier, and a data tier. MDM provides an additional tier in between the business tier and the data tier. Data is provided to the business logic tier through MDM as a service. In providing this service, MDM improves the quality of the master data provided to the enterprise when compared to a situation where this data would have been accessed directly by the business logic.

As you will see, there is a lot of business value in doing so. In addition, granting access to master data through services allows seamless integration of master data in service orchestration to run business processes. Furthermore, encapsulation of master data into services allows consistent enforcement of duplicate prevention and data standardization, which, as a side effect, establishes trust in your master data.

### 2.4.1 MDM as an SOA Enabler

This discussion leads us back to the very important point about why you need MDM to enable SOA that we started this chapter with. Early on, SOA technology focused mainly on mapping service components to business processes, and on mapping business processes to business components. However, in many cases the quality of the data involved was ignored or merely an afterthought. Implementing services on top of distinct systems and repositories, each with their own levels of data quality and without cross-system de-duplication, leads to SOA systems with poor data quality characteristics. How can one, for example, implement a single-enterprise *getCustomer* SOA service if parts of the customer data are spread out over several systems with different levels of data consistency, and without proper data matching and de-duplication? Without MDM, applying an SOA can actually make matters regarding information worse instead of better, because low-quality data would now be exposed through untrusted services and thus consumed by even more consumers. Many SOA projects have failed because of this lack of data consistency across the enterprise.

MDM offers a new level of data quality that assists SOA enablement. Exactly those objects that occur frequently in the enterprise and are of high importance to many business processes—the core business entities—fall under the master data umbrella. The services offered by the MDM System ensure the quality[24] and timeliness of the master data and assist other SOA projects to achieve their enterprise objectives. The MDM services offer a consistent view of the master data within a context and with proper security, visibility, and entitlement constraints. MDM provides the technical foundation for master data use in an SOA-style architecture.

24. See Chapter 1 for a description of different data quality aspects.

### 2.4.2 MDM without SOA

It is possible to deliver MDM without services and without an SOA architecture. Instead of providing services to maintain and retrieve the master data, data extraction, transformation, and loading tools (ETL) could be used to build an MDM infrastructure. Data can be validated, standardized, cleansed, and de-duplicated while it is being loaded into an MDM data model. However, the ETL jobs to build such an infrastructure would quickly become fairly complicated in any approach to the same level of data quality as can be achieved by a set of well-designed MDM SOA services. In a rather simple scenario, such a non-SOA MDM Solution might work, but it would suffer from severe limitations caused by the lack of real-time SOA services. It would, for example, be very hard to integrate the operations that such a solution offers into larger processes in the way an SOA service could participate in a larger service. Capabilities such as instant updates with proper transactional (two-phased commit, XA compliant) control would not be available. Security, data visibility, and data entitlements would be hard to control if data access is done directly to the database, and so forth. In short, not only does MDM serve as an SOA enabler, it also requires SOA to reach its full potential.

### 2.4.3 MDM and Evolvability, Flexibility, and Adaptability

If the business ecosystem were static, every required system would have been built by now and IT staff would have been sent home a decade ago. Fortunately for those folks, the only constant in business is change. Everything changes: Consumer markets change, labor markets change, competitors change, technology changes, the geopolitical environment changes, law and regulations change, mergers and acquisitions happen, and so on, and all these influence the business environment. The rate of change has in fact been accelerating in recent years. An increasing number of companies need to outsource parts of their operations to stay competitive; they might need to reallocate parts of their business to other regions, share value chains with partner companies, and adapt to new rules and regulations. An enterprise that cannot adapt to such changes in a short enough time frame will be left behind and miss significant opportunities. But in many cases change can actually create opportunities as long as the business can adapt to them: New channels and markets can open up, cheaper manufacturing options become available, and new consumer needs arise. IT needs to support the business and adapt to these changes as well, and in other cases it needs to drive the change to move the business forward.

MDM plays a central role in enabling the ability to change and to evolve by centralizing the maintenance and governance of the master data. Changes to the business typically affect the master data elements of an enterprise. The impact of these changes on the master data can be determined more accurately in a managed model using MDM, and the number of touch points where changes need to be applied is reduced when MDM is used. In this way, an MDM System can improve the responsiveness to business changes by centrally changing the way master data is maintained. An example of this would be new requirements around privacy regulations. Enabling privacy information, to be held at a customer level, in all of the systems that maintain customer information in an unmanaged master data environment is cumbersome and, because of resulting synchronization requirements, also error-prone.

Using the MDM System to track such new privacy information at an enterprise level for customer master data reduces the impact of this extra requirement to the business significantly.

MDM extends beyond managing the master data solely for the purpose of enabling a decoupling of information from their original business process and applications. It also provides flexibility for changes to business processes, laws, and regulations. Master data that is no longer trapped within the confines of Line of Business, stovepipe applications can venture beyond the limitations of the business processes of that system. Additional business processes can be defined, and validations, augmentations, and changes can be applied. Therefore, the design of the MDM Solution architecture should be based upon service-oriented design principles and SOA concepts. Here, information is packaged as a service to business processes, so that consistent, manageable information is made available to business processes in a standardized way that enables reuse. Changes to the service implementations are hidden from clients through the use of SOA. The design of the MDM System should enable the composition and extension of these MDM services and schema in a way that provides stability moving forward. It should allow for the addition of new entities or attributes to the master data and for changes to their definition. It should enable the creation or composition of new services and allow for changes to be made to the behavior of the existing ones. The MDM System needs to enable the ability to rapidly respond to business-driven and legislated changes and provide the ability to improve business decisions and capabilities with up-to-date information at a global level.

An enterprise's business grows over time, and with growth come new requirements in the area of scalability of the MDM System. Typical areas of growth for an MDM System are:

- Connect to more application systems, with greater heterogeneity, downstream or upstream.[25] This can require additional adaptors, data flows, message queues, and so on.
- Support a larger user community, with additional types of users. Wider adoption of MDM usage within the enterprise leads, for example, to additional lines of business starting to use the system. A larger user community often requires additional user roles, additional authorization rules, and so on.
- Handle a larger data volume, possibly federated data across multiple databases.
- Keep more historical data or versioned data.
- Handle larger transaction volumes.
- Handle additional delivery channels: Web, self-service kiosk, IVR, mobile users.
- Include additional and more complex workflow flows.
- Deploy additional SOA services. Build more complex SOA services, for example, by building additional composite services.
- Handle more MDM domains (party, product, account, location, etc.).
- Support additional business processes, possibly through integration with business process software.
- Handle more data extensions and additions to the base domains.

25. *Downstream:* For example, exports or notifications originating from the MDM System. *Upstream:* For example, systems sending requests or changes to the MDM System.

- Handle increasingly complex business rules, such as those around de-duplication.
- Handle larger delta feeds via batch, such as daily updates from other systems.
- Handle increased reporting requirements.
- Handle more complex UIs, covering more user roles.
- Support multinational deployment.
- Higher service level requirements, increased up-time of the system.

Some of the infrastructure changes that can be made to support such growth are:

- Support more nodes in an application server cluster.
- Use partitioned databases.
- Change deployment topologies such as from UNIX®-based to mainframe-based.
- Deploy an ESB or other form of middleware.
- High-availability configurations.

Another way in which the MDM System needs to be able to evolve is in its technical architecture. Like business, technology constantly changes, and to increase the longevity of an MDM Solution and to lower its Total Cost of Ownership (TCO), it needs to be able to adapt to newer technologies. Many of the SOA principles we discussed earlier, like modularity, loose coupling, and componentization aid in this area. This technical evolvability needs to occur on these different levels:

1. An implementation of an MDM System for a specific enterprise or scenario needs to be able to migrate relatively smoothly to newer versions of the core MDM System software. This is not a trivial requirement. Configurations, extensions, and additions of a commercial MDM System are inevitable in a real-life enterprise situation. What is critical, though, is that those modifications survive an MDM System software upgrade without incurring large amounts of rework. This mandates the presence of frameworks, migration tools, metadata, and development tooling to support such upgrades.
2. The second form of technical evolvability is that of the underlying technology on which the MDM System is based. Specifications such as J2EE and Web Services evolve over time, and they do so at a much faster rate than the technologies that were used for enterprise predecessors such as COBOL or C++. Moreover, new promising technologies pop up constantly, and the MDM System software needs to maintain its technical currency. Its interaction points need to be able to handle new types of protocols and communication such as REST or RSS. Its underlying persistence mechanism needs to be abstracted so it can be replaced with improved technology. Its plug-in points, such as those for data standardization, need to be flexible enough to handle new technologies.
3. MDM software systems typically include a stack of supporting infrastructure components such as a database management system, message middleware, integration software, and an application server. None of these components are exclusively used by the MDM System but typically represent components of the enterprise architecture. They all evolve over time—newer hot fixes, fixpacks, and versions are released and new features are added to these products. To stay on a supported platform, the MDM System needs to stay in step with these changes. There are two major ways

to do this. The first, and simplest, way is to migrate the MDM product over to the new stack without any changes and to test it there. This typically only offers a minimal level of benefit to the end user. The second is to actually take advantage of new features offered in the new stack components, which will allow for new functionality based on new features in the underlying stack software. Obviously, the second approach is more complex and time-consuming, but it also provides more benefits.

In selecting or building an MDM Solution for an enterprise, one has to ensure that the system is both able to handle the present requirements and equipped to scale or transform in all of the required manners described here. This sounds easier than it is in reality. The pressures to deliver a tactical measure of immediate value and success from an MDM deployment in the short term sometimes overshadow the strategic, longer-term vision of a scalable, highly available, and extensible MDM architecture.

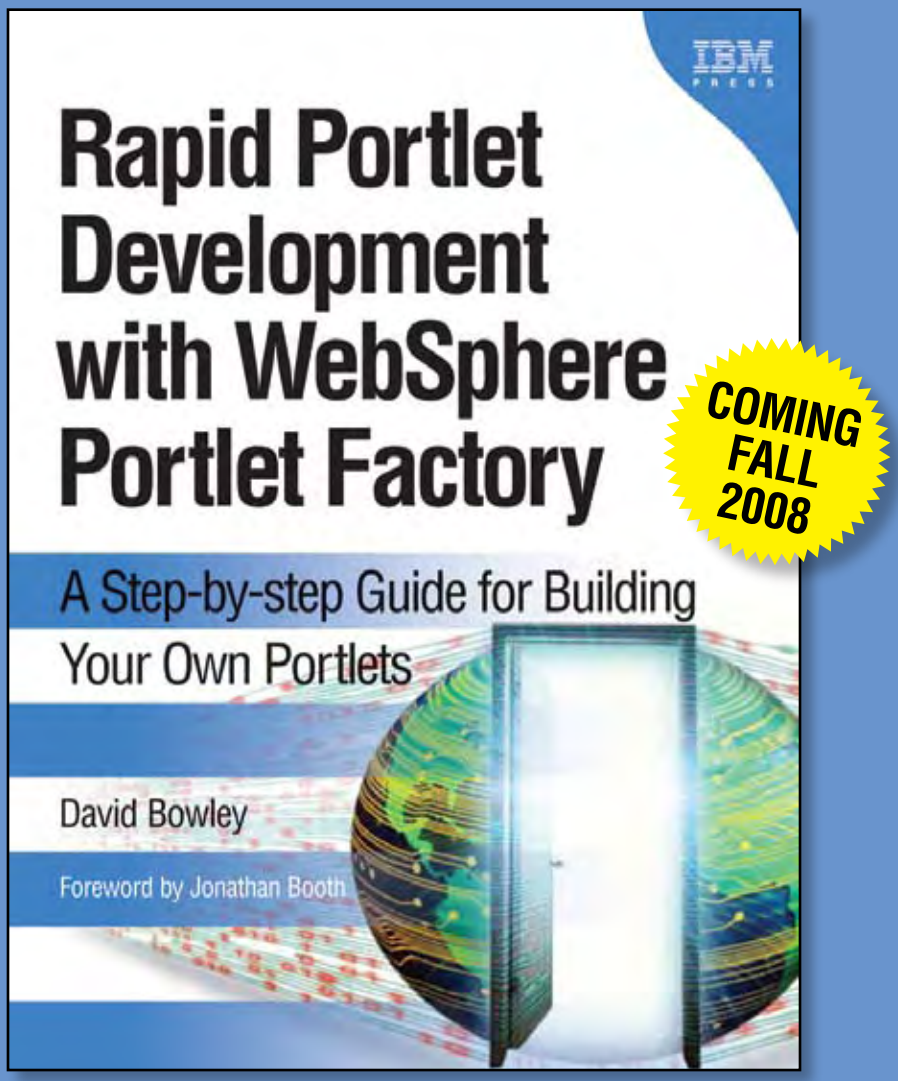
## Conclusion

In this chapter, we introduced key concepts of Service-Oriented Architecture. We showed how MDM can enable consistent use of key business entities such as customer or product in SOA. Without MDM, applying SOA can make matters regarding information worse—not better—because low-quality data would be exposed through untrusted services and thus consumed by even more consumers. With this insight, you can see now how MDM provides the technical foundation for master data use in an SOA-style architecture. From an SOA standpoint, the service concept applied to information leads to the discovery of the concept of information as a service. Accessing master data is not directly done through database interfaces, as you will see in Chapter 3 when we explain the MDM Reference Architecture. Instead, access to master data uses services encapsulating a lot of functionality, such as duplicate checking or address standardization. Thus, master data access instantiates the information as a service concept—in other words, master data services are one type of information service. In the next chapter, we dive into the architectural details of the MDM Reference Architecture, which is followed by a more detailed discussion about MDM security and privacy, in Chapter 4.

## References

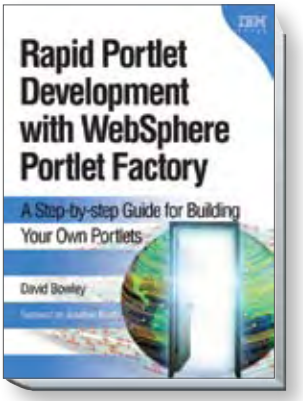
1. Andrew White, Charles Abrams, Gartner Group 30, March 2005. *Service-Oriented Business Applications Require EIM Strategy*. ID Number G00124926.
2. N. Bieberstein, S. Bose, M. Fiammante, K. Jones, R. Shaw, April 2006. *Service Oriented Architecture (SOA) Compass—Business Value, Planning, and Enterprise Roadmap*. IBM Press, Pearson plc.
3. Lublinsky, B. Jan 2007. *Defining SOA as an architectural style*. Retrieved March 2008 from IBM developerWorks®: http://www.ibm.com/developerworks/library/ar-soastyle/index.html#resources.
4. Rob High, Jr., Stephen Kinder, Steve Graham, November 2005. *IBM's SOA Foundation, An Architectural Introduction and Overview, An IBM Whitepaper*. http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-whitepaper.pdf.

5. Ali Arsanjani, November 9. *Service oriented modeling and architecture*. Retrieved March 2008 from IBM developerWorks: http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/.
6. Carol Baroudi and Dr. Fern Halper, Hurwitz & Associates. Feb 27, 2007. *Executive Survey: SOA Implementation Satisfaction*. Retrieved March 2008 from cio.co.uk: http://www.cio.co.uk/cmsdata/whitepapers/4474/110206_Mindreef_Report[1][1].pdf.
7. David Chappell, *SOA and the Reality of Reuse*. Retrieved March 2008 from Opinari: http://www.davidchappell.com/HTML_email/Opinari_No16_8_06.html.
8. Joe McKendrick, *Ten companies where SOA made a difference in 2006*. Retrieved March 2008 from ZDNet: http://blogs.zdnet.com/service-oriented/?p=781.
9. Anthony Bradley, *SOA and Modularity, Net-Centricity, SOA, and Web 2.0*. Retrieved March 2008 from TypePad.com: http://ajbradley.typepad.com/soa_and_netcentricity/2006/08/soa_and_modular.html.
10. Martin Keen, Andrea Ames, Asit Dan, Robert A. Dickson, Simon Harris, Arthur Kaufmann, Ivan Milman, Guenter Sauter, Mahesh Viswanathan. *Case Study: Information as a Service SOA Scenario, IBM Redbook*. Retrieved March 2008 from IBM Red Books: http://www.redbooks.ibm.com/redpapers/pdfs/redp4382.pdf.
11. Larry P. English, 1999. *Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits*. John Wiley & Sons.
12. Thomas C. Redman, PhD, 2001. *Data Quality: The Field Guide*. Digital Press.



David Bowley

SIGN UP FOR IBM Press Newsletter




# Rapid Portlet Development with WebSphere Portlet Factory

## A Step-by-step Guide for Building Your Own Portlets

©2008 ISBN:0-13-713446-0

In *Rapid Porlet Development with WebSphere Portlet Factory,* author David Bowley presents a series of independent walkthroughs, each based on solving the most common scenarios that come up in portlet development. Along the way this step-by-step guide includes useful tips, tricks, shortcuts, and previously undocumented "gotchas." The many practical examples are offered online so you can simply plug the examples straight into your own development projects and get tangible results immediately.

COMING FALL 2008

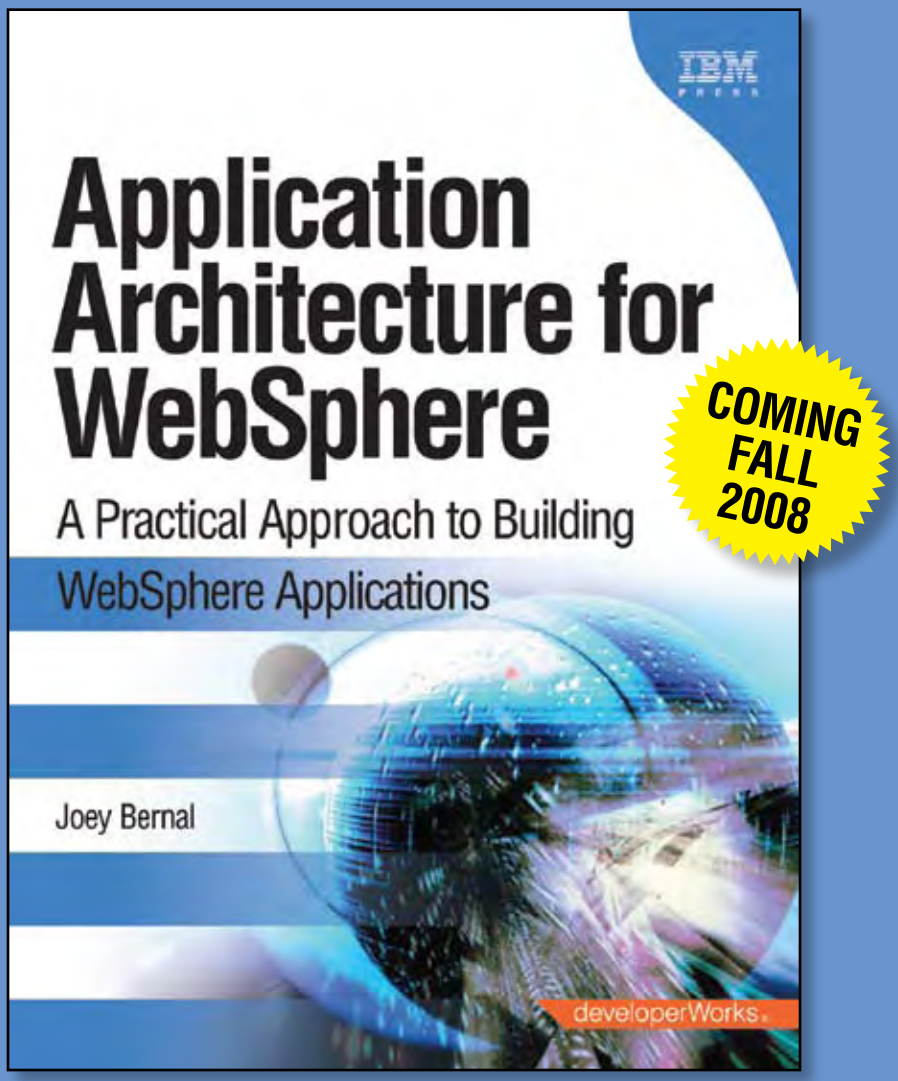
### Table of Contents



### About the Author

**DAVID BOWLEY** is a consultant for e-Centric Innovations, an e-business solution provider in Melbourne, Australia. Over the last nine years, David has worked extensively with IBM technologies and has been involved in some of Australia's largest portal projects. David is a frequent contributor to various technical magazines, including *The View* and *Lotus Advisor.*

IBM Press™



Joey Bernal

SIGN UP FOR IBM Press Newsletter


# Application Architecture for WebSphere

## A Practical Approach to Building WebSphere Applications

©2008 ISBN: 0-13-712926-2

*Application Achitecture for WebSphere* provides architects or teams with recommendations for designing and adhering to a layered architecture approach for WebSphere applications. The author offers sample architectures from several different organizations; these examples grow in complexity as new layers are added and modified during the progression of the book.

### Table of Contents



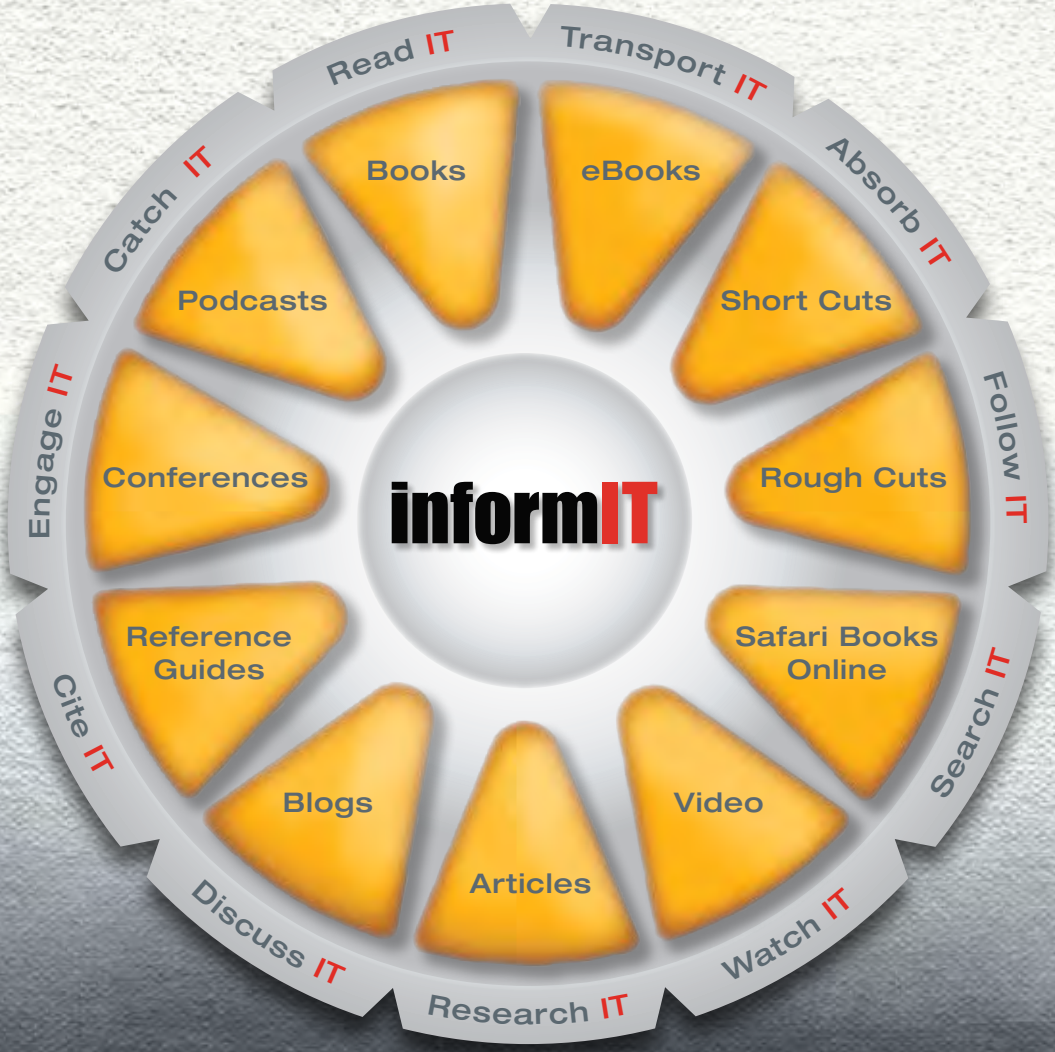
COMING FALL 2008

### About the Author

**JOEY BERNAL** is an Executive IT Specialist with Software Services for Lotus, and a member of the IBM WebSphere Portal Tech Team. Sr. Certified with IBM as an IT Specialist, he has an extensive background in the design and development of Portal and Web Applications. Joey leads the Software Services team approach for assisting clients with their cross-brand challenges that leverage WebSphere Portal.

IBM Press™

# LearnIT at InformIT




## 11 WAYS TO LEARN IT AT www.InformIT.com/learn