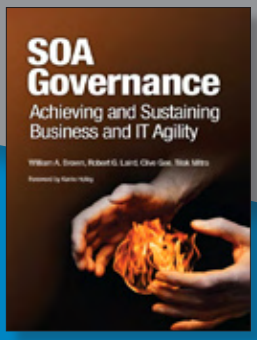
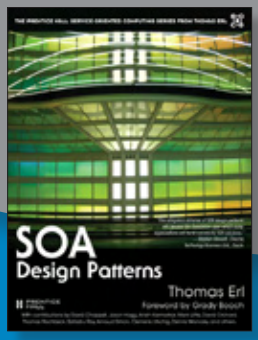
# SOA eBook

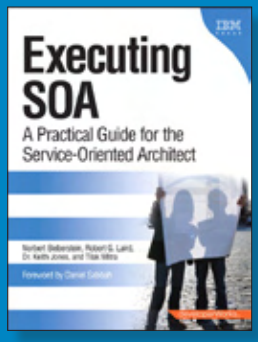
## Patterns, Mashups, Governance, Service Modeling, and More

### SAMPLE CHAPTERS






informit.com/learnSOA

informIT.com
THE TRUSTED TECHNOLOGY LEARNING SOURCE

Addison Wesley | PRENTICE HALL | IBM Press

# WHAT PEOPLE ARE SAYING


"I recommend *Service-Oriented Architecture: Concepts, Technology, and Design* to anyone involved with designing and implementing an SOA initiative within an enterprise."

– ART SEDIGHI


"This book is a great reference for large scale organizations that need a handy reference to the various possible processes and artifacts that could comprise an extensive SOA Governance process."

– KELVIN MEEKS


"Adoption of these proven patterns should help in addressing the quality and capabilities of SOA efforts."

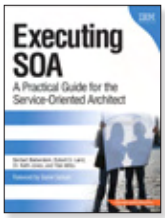
– JOE McKENDRICK


"Enterprise architects think in high-level architectural patterns. This book cuts through the mashup marketing fluff and gets right down to presenting patterns as a way to analyze and solve enterprise problems using mashups."

– JOHN CRUPI, Chief Technology Officer of JackBe and Coauthor of *Core J2EE Patterns*


"…*SOA Principles of Service Design* should appeal to anyone who is interested in doing SOA, regardless of their choice of implementation technology."

– DMITRI ILKAEV




"*Executing SOA* is not a book that should take up space on your bookshelf. It should be next to your keyboard for the duration of your SOA project."

– ASLAM KHAN, Architect Zone, DZone.com


"Thomas Erl is on this list three times (#6, #17, #24). He didn't pay me for that, unfortunately. He just did it all by himself."

– Special distinction on Jurgen Appelo's listing of the "Top 50 New Software Development Books"

"…*Web Service Contract Design and Versioning for SOA* is a detailed, in-depth reference…"

– INTERNET BOOKWATCH


"…a great advocacy book on why and how and where businesses need SOA, covering a lot of the bases of business thinking."

– JOE McKENDRICK

## THESE BOOKS ARE AVAILABLE AT BOOKSTORES INCLUDING:

amazon.com. BARNES&NOBLE www.bn.com BORDERS International Resellers

Visit **informit.com/learnsoa** for new releases, excerpts, and special offers.

# SOA eBook

## Patterns, Mashups, Governance, Service Modeling, and More

### eBOOK TABLE OF CONTENTS

- On SOA Podcast Channel LEARN MORE...


**Service-Oriented Architecture**
9780131858589
Thomas Erl
CHAPTER 3: Introducing SOA


**SOA Governance**
9780137147465
William A. Brown, Robert G. Laird, Clive Gee, Tilak Mitra
CHAPTER 8: SOA Governance Case Study


**SOA Design Patterns**
9780136135166
Thomas Erl
CHAPTER 12: Service Implementation Patterns


**Mashup Patterns**
9780321579478
Michael Ogrinz
CHAPTER 1: Understanding Mashup Patterns


**SOA Principles of Service Design**
9780132344821
Thomas Erl
CHAPTER 5: Understanding Design Principles


**Executing SOA**
9780132353748
Norbert Bieberstein, Robert G. Laird, Dr. Keith Jones, Tilak Mitra
CHAPTER 4: A Methodology for Service Modeling and Design


**Web Service Contract Design and Versioning for SOA**
9780136135173
Thomas Erl, Anish Karmarkar, Priscilla Walmsley, Hugo Haas, L. Umit Yalcinalp, Ph.D., Canyang Kevin Liu, David Orchard, Andre Tost, James Pasley
CHAPTER 3: SOA Fundamentals and Web Service Contracts


**The New Language of Business**
9780131956544
Sandy Carter
CHAPTER 11: Putting It All Together

- More SOA Books of Interest LEARN MORE...
- Safari Books Online LEARN MORE...
- InformIT LEARN MORE...
- Learn SOA Web Page LEARN MORE...

BROUGHT TO YOU BY





UPPER SADDLE RIVER, NJ | BOSTON | INDIANAPOLIS | SAN FRANCISCO | NEW YORK | TORONTO | MONTREAL | LONDON | MUNICH
PARIS | MADRID | CAPETOWN | SYDNEY | TOKYO | SINGAPORE | MEXICO CITY

# SOA Podcasts – Conversations with Thought Leaders in SOA


**InformIT presents weekly SOA technology podcasts featuring the world's leading experts and best-selling authors.**

LISTEN AT informIT.com

SUBSCRIBE AT iTunes

### Service Contract-Related SOA Design Patterns

BY THOMAS ERL

Click to listen

Joe McKendrick asks best-selling author Thomas Erl to explain some of the SOA design patterns that address service contract design issues that occur both during the initial service design stage as well as subsequent to deployment, including versioning.

### Executing SOA: SOA Governance

BY ROBERT LAIRD

Click to listen

Robert Laird, co-author of *Executing SOA* and *SOA Governance,* provides insights on Service-Oriented Architecture and SOA Governance. Learn about an SOA Governance paradigm & gain expert guidance for successfully delivering on even the largest and most complex SOA initiatives.

### XML Schemas and SOA

BY PRISCILLA WALMSLEY

Click to listen

*Web Service Contract Design & Versioning for SOA* co-author Priscilla Walmsley talks about how XML Schemas are used in SOA, how to build XML Schemas to better support service contracts, designs, and important "gotchas" to look out for.

### The New Language of Business: SOA and Web 2.0

BY SANDY CARTER

Click to listen

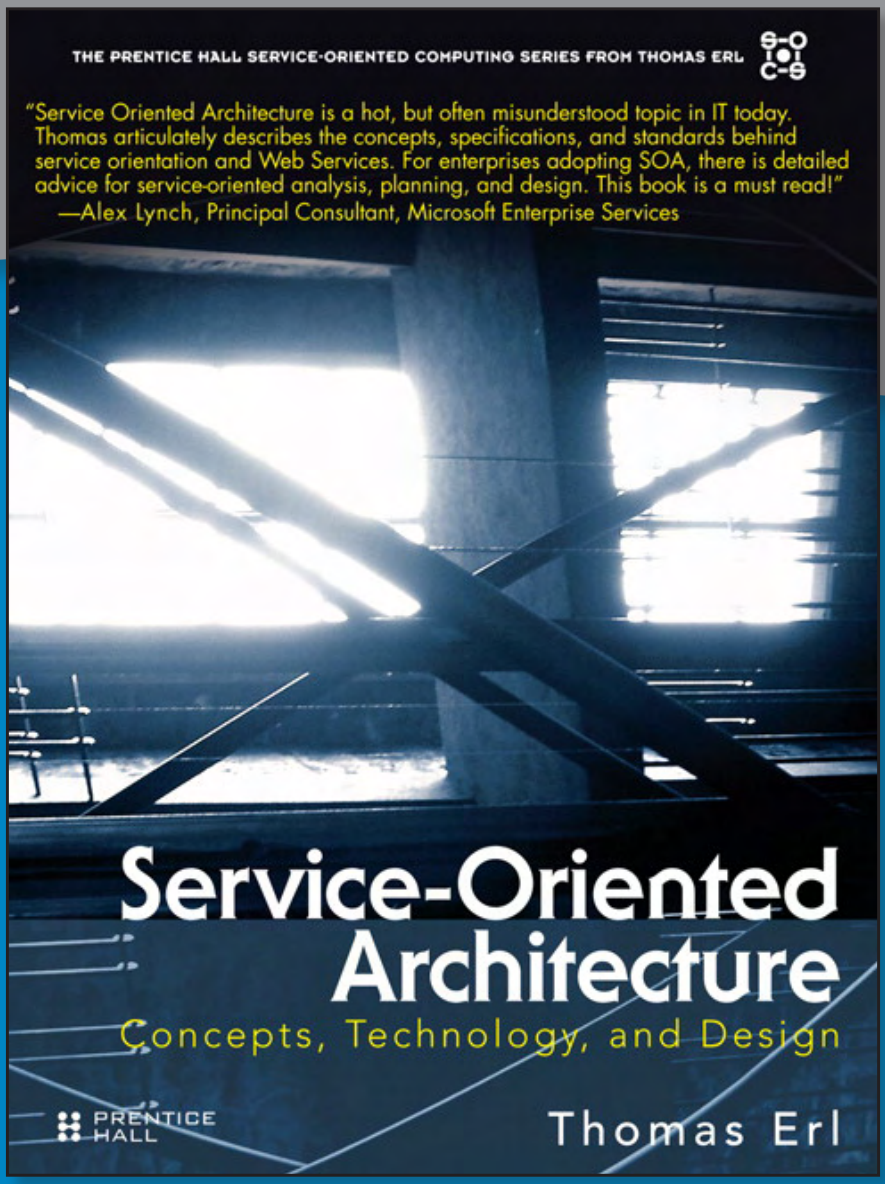
Sandy Carter, VP of SOA & WebSphere Strategy, Channels & Marketing at IBM, discusses the relationship between SOA and Web 2.0 in Part 1 of this 3 part series.

On Podcasts

informIT.com THE TRUSTED TECHNOLOGY LEARNING SOURCE

Addison-Wesley Cisco Press EXAM/CRAM IBM Press. QUE PRENTICE HALL SAMS

Safari Books Online



Thomas Erl

BUY ME


# Service-Oriented Architecture

## Concepts, Technology, and Design

### The foremost "how-to" guide to SOA

Service-Oriented Architecture (SOA) is at the heart of a revolutionary computing platform that is being adopted world-wide and has earned the support of every major software provider. In Service-Oriented Architecture: Concepts, Technology, and Design, Thomas Erl presents the first end-to-end tutorial that provides step-by-step instructions for modeling and designing service-oriented solutions from the ground up.

Erl uses more than 125 case study examples and over 300 diagrams to illuminate the most important facets of building SOA platforms: goals, obstacles, concepts, technologies, standards, delivery strategies, and processes for analysis and design.

HIS BOOK'S BROAD COVERAGE INCLUDES.

- Detailed step-by-step processes for service-oriented analysis and service-oriented design
- An in-depth exploration of service-orientation as a distinct design paradigm, including a comparison to object-orientation
- A comprehensive study of SOA support in .NET and J2EE development and runtime platforms
- Descriptions of over a dozen key Web services technologies and WS-* specifications, including explanations of how they interrelate and how they are positioned within SOA
- The use of "In Plain English" sections, which describe complex concepts through non-technical analogies
- Guidelines for service-oriented business modeling and the creation of specialized service abstraction layers
- A study contrasting past architectures with SOA and reviewing current industry influences
- Project planning and the comparison of different SOA delivery strategies

The goal of this book is to help you attain a solid understanding of what constitutes contemporary SOA along with step-by-step guidance for realizing its successful implementation.

AVAILABLE
- BOOK: 9780131858589
- SAFARI ONLINE Safari Books Online
- KINDLE: B000OZ0NBC

## About the Author

THOMAS ERL is the world's top-selling SOA author, *Series Editor of the Prentice Hall Service-Oriented Computing Series* from Thomas Erl, and Editor of The *SOA Magazine* (www.soamag.com).With over 100,000 copies in print world-wide, his books have become international bestsellers and have been formally endorsed by senior members of major software organizations, such as IBM, Microsoft, Oracle, BEA, Sun, Intel, SAP, CISCO, and HP. Thomas is the founder of SOA Systems Inc. and the internationally recognized SOA Certified Professional program (www.soacp.com). Articles and interviews by Thomas have been published in numerous publications, including *The Wall Street Journal* and *CIO Magazine.*

PRENTICE HALL

informit.com/ph

# Chapter 3

# Introducing SOA

Before we get into the many details of understanding and building service-oriented solutions, let's first introduce some basic concepts and establish a range of associated issues.

> **NOTE**
>
> Be sure to also visit `www.whatissoa.com` for a concise overview of SOA and service-oriented computing and descriptions of the common strategic goals and benefits.

## 3.1 Fundamental SOA

Because the term "service-oriented" has existed for some time, it has been used in different contexts and for different purposes. One constant through its existence has been that it represents a distinct approach for separating concerns. What this means is that logic required to solve a large problem can be better constructed, carried out, and managed if it is decomposed into a collection of smaller, related pieces. Each of these pieces addresses a concern or a specific part of the problem.

This approach transcends technology and automation solutions. It is an established and generic theory that can be used to address a variety of problems. What distinguishes the service-oriented approach to separating concerns is the manner in which it achieves separation.

### 3.1.1 A service-oriented analogy

Let's take your average cosmopolitan city. It is already full of service-oriented businesses. Individual companies are service-oriented in that each provides a distinct service that can be used by multiple consumers. Collectively, these businesses comprise a business community. It makes sense for a business community not to be served by a single business outlet providing all services. By decomposing the community into specialized, individual outlets, we achieve an environment in which these outlets can be distributed.

When coupled with "architecture," service-orientation takes on a technical connotation. "Service-oriented architecture" is a term that represents a model in which automation

logic is decomposed into smaller, distinct units of logic. Collectively, these units comprise a larger piece of business automation logic. Individually, these units can be distributed.

Distributing automation logic into separate units is nothing new. What is it then that makes *service-oriented* separation so different? Much of this book is dedicated to answering that question. However, let's take a preliminary look at some notable distinctions.

Even in a distributed business community, if we impose overbearing dependencies, we could inhibit the potential of individual businesses. Although we want to allow outlets to interact and leverage each other's services, we want to avoid a model in which outlets form tight connections that result in constrictive inter-dependencies. By empowering businesses to self-govern their individual services, we allow them to evolve and grow relatively independent from each other.

Though we encourage independence within our business outlets, we must still ensure that they agree to adhere to certain baseline conventions—for example, a common currency for the exchange of goods and services, a building code that requires signage to conform to certain parameters or perhaps a requirement that all employees speak the same language as the native consumers. These conventions standardize key aspects of each business for the benefit of the consumers without significantly imposing on the individual business's ability to exercise self-governance.

Similarly, service-oriented architecture (SOA) encourages individual units of logic to exist autonomously yet not isolated from each other. Units of logic are still required to conform to a set of principles that allow them to evolve independently, while still maintaining a sufficient amount of commonality and standardization. Within SOA, these units of logic are known as *services*.

### 3.1.2 How services encapsulate logic

To retain their independence, services encapsulate logic within a distinct context. This context can be specific to a business task, a business entity, or some other logical grouping.

The concern addressed by a service can be small or large. Therefore, the size and scope of the logic represented by the service can vary. Further, service logic can encompass the logic provided by other services. In this case, one or more services are composed into a collective.

For example, business automation solutions are typically an implementation of a business process. This process is comprised of logic that dictates the actions performed by the solution. The logic is decomposed into a series of steps that execute in predefined sequences according to business rules and runtime conditions.

As shown in Figure 3.1, when building an automation solution consisting of services, each service can encapsulate a task performed by an individual step or a sub-process comprised of a set of steps. A service can even encapsulate the entire process logic. In the latter two cases, the larger scope represented by the services may encompass the logic encapsulated by other services.

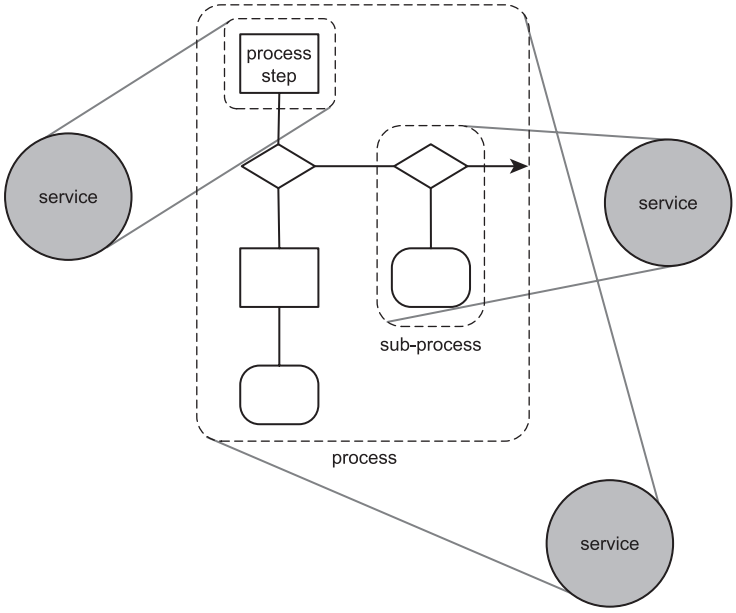



**Figure 3.1**
Services can encapsulate varying amounts of logic.

For services to use the logic they encapsulate they can participate in the execution of business activities. To do so, they must form distinct relationships with those that want to use them.

### 3.1.3 How services relate

Within SOA, services can be used by other services or other programs. Regardless, the relationship between services is based on an understanding that for services to interact, they must be aware of each other. This awareness is achieved through the use of *service descriptions*.

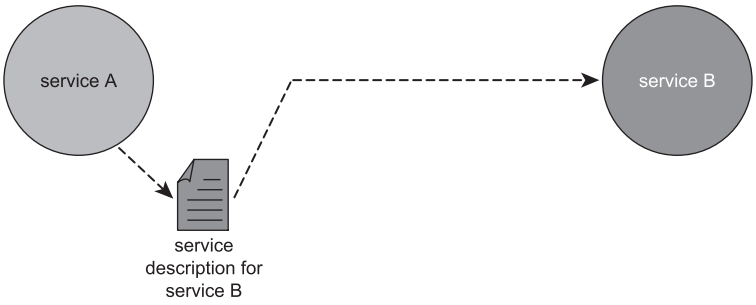



**Figure 3.2**
Because it has access to service B's service description, service A has all of the information it needs to communicate with service B.

A service description in its most basic format establishes the name of the service and the data expected and returned by the service. The manner in which services use service descriptions results in a relationship classified as *loosely coupled*. For example, Figure 3.2 illustrates that service A is aware of service B because service A is in possession of service B's service description.

For services to interact and accomplish something meaningful, they must exchange information. A communications framework capable of preserving their loosely coupled relationship is therefore required. One such framework is *messaging*.

### 3.1.4 How services communicate

After a service sends a message on its way, it loses control of what happens to the message thereafter. That is why we require messages to exist as "independent units of communication." This means that messages, like services, should be autonomous. To that effect, messages can be outfitted with enough intelligence to self-govern their parts of the processing logic (Figure 3.3).

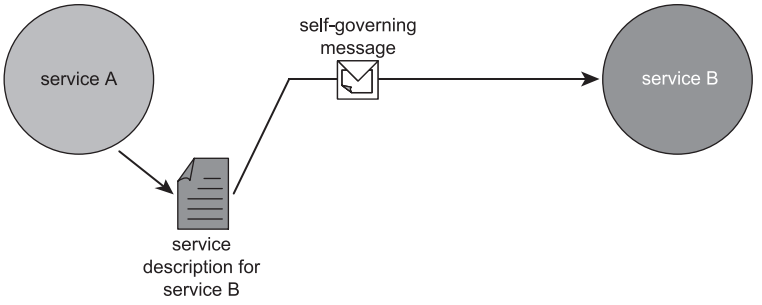



**Figure 3.3**
A message existing as an independent unit of communication.

Services that provide service descriptions and communicate via messages form a basic architecture. So far, this architecture appears similar to past distributed architectures that support messaging and a separation of interface from processing logic. What distinguishes ours is how its three core components (services, descriptions, and messages) are designed. This is where *service-orientation* comes in.

### 3.1.5 How services are designed

Much like object-orientation, service-orientation has become a distinct design approach which introduces commonly accepted principles that govern the positioning and design of our architectural components (Figure 3.4).

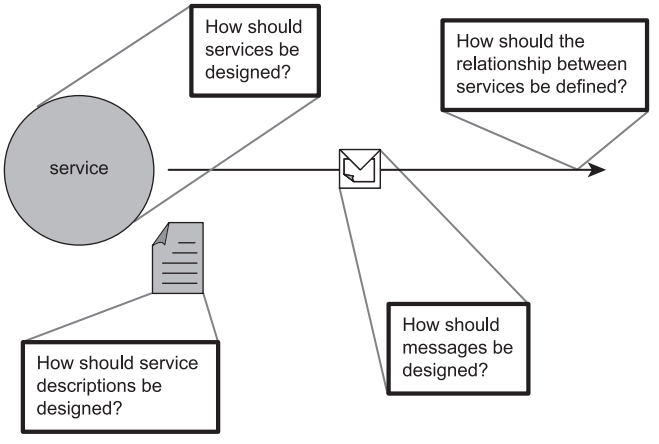



**Figure 3.4**
Service-orientation principles address design issues.

The application of service-orientation principles to processing logic results in standardized service-oriented processing logic. When a solution is comprised of units of service-oriented processing logic, it becomes what we refer to as a *service-oriented solution*.

The individual principles of service-orientation are fully explained later in this book. For the purpose of providing a preliminary introduction, let's highlight some of the key aspects of these principles here:

- *Loose coupling*—Services maintain a relationship that minimizes dependencies and only requires that they retain an awareness of each other.
- *Service contract*—Services adhere to a communications agreement, as defined collectively by one or more service descriptions and related documents.
- *Autonomy*—Services have control over the logic they encapsulate.
- *Abstraction*—Beyond what is described in the service contract, services hide logic from the outside world.
- *Reusability*—Logic is divided into services with the intention of promoting reuse.
- *Composability*—Collections of services can be coordinated and assembled to form composite services.
- *Statelessness*—Services minimize retaining information specific to an activity.
- *Discoverability*—Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

With a knowledge of the components that comprise our basic architecture and a set of design principles we can use to shape and standardize these components, all that is missing is an implementation platform that will allow us to pull these pieces together to build service-oriented automation solutions. The *Web services* technology set offers us such a platform.

### 3.1.6 How services are built

As we mentioned earlier, the term "service-oriented" and various abstract SOA models existed before the arrival of Web services. However, no one technology advancement has been so suitable and successful in manifesting SOA than Web services.

All major vendor platforms currently support the creation of service-oriented solutions, and most do so with the understanding that the SOA support provided is based on the use of Web services. Therefore, while we fully acknowledge that achieving SOA does not require Web services, this book's focus is on how SOA can and should be realized through the use of the Web services technology platform.

### 3.1.7 Primitive SOA

The past few sections have described the individual ingredients for what we call *primitive SOA*. It is labeled as such because it represents a baseline technology architecture that is supported by current major vendor platforms.

All forms of SOA we explore from here on are based on and extend this primitive model. Some of the extensions we discuss are attainable today through the application of advanced design techniques, while others rely on the availability of pre-defined Web services specifications and corresponding vendor support.

**CASE STUDY**

RailCo's accounting solution exists as a two-tier client-server application, where the bulk of application logic resides within an executable deployed on client workstations. The details of this application are described in the next chapter. For now, let's identify two of the primary tasks this application can perform:

- Enter Customer Purchase Order
- Create Customer Order

The completion of each task involves a series of steps that constitute a business process. This process was originally modeled using standard workflow logic and then implemented as part of a packaged solution. Within the application, the process may or may not be separately represented by individual sets of programming routines. Regardless, it is compiled into a single executable that provides a fixed manner in which the process is automated.

Within a service-oriented business model, the logic behind each process would be partitioned into one or more services. If multiple services are used, the execution of the entire process would involve the composition of these services. In this case, each service may represent a sub-process or even a single step within the process that can be executed independently. For example, the Create Customer Order Process may consist of the following sub-processes:

- Retrieve Purchase Order Data
- Check Inventory Availability
- Generate Back Order
- Publish Customer Order

As we know, a process in its entirety can be viewed and modeled as a service. Additionally, one or more processes can be combined to represent an even larger service. For example, the Create Customer Order and Generate Customer Invoice Processes may be combined to form a single Order Processing Process.

Finally, we would also expect these processes to be flexible so that they can incorporate processes or resources that exist elsewhere in the enterprise. For instance, we may decide to extend the Order Processing Process to include a sub-process that automatically retrieves the customer's current accounts payable mailing address. This sub-process may already exist as part of a separate Customer Contact Reporting Process.

To implement such a model, we need a technical architecture capable of providing the following:

- The ability for business automation logic to be partitioned into units so as to properly represent services.
- The ability for these units of logic to be relatively independent of each other so as to support the requirement for them to participate in different compositions.
- The ability for these units of logic to communicate with each other in such a manner that their respective independence is preserved.

The fundamental characteristics of service encapsulation, loose-coupling, and messaging, as realized through service-orientation principles and the Web services technology set, collectively fulfill these requirements through the implementation of a primitive SOA.

**SUMMARY OF KEY POINTS**

- SOA and service-orientation are implementation-agnostic paradigms that can be realized with any suitable technology platform.
- Our primitive SOA model represents a mainstream variation of SOA based solely on Web services and common service-orientation principles.
- Throughout the remainder of this book, any reference to the term "SOA" implies the primitive SOA model.

## 3.2 Common characteristics of contemporary SOA

Numerous recent and ongoing industry trends and developments have shaped the real world look of SOA. Its founding principles remain, but many have been expanded primarily because the opportunity to do so has been readily acted upon.

Major software vendors are continually conceiving new Web services specifications and building increasingly powerful XML and Web services support into current technology platforms. The result is an extended variation of service-oriented architecture we refer to as *contemporary SOA*.

Contemporary SOA builds upon the primitive SOA model by leveraging industry and technology advancements to further its original ideals. Though the required implementation technology can vary, contemporary SOAs have evolved to a point where they can be associated with a set of common characteristics.

Specifically, we explore the following primary characteristics:

- Contemporary SOA is at the core of the service-oriented computing platform.
- Contemporary SOA increases quality of service.
- Contemporary SOA is fundamentally autonomous.
- Contemporary SOA is based on open standards.
- Contemporary SOA supports vendor diversity.
- Contemporary SOA fosters intrinsic interoperability.
- Contemporary SOA promotes discovery.
- Contemporary SOA promotes federation.
- Contemporary SOA promotes architectural composability.
- Contemporary SOA fosters inherent reusability.
- Contemporary SOA emphasizes extensibility.
- Contemporary SOA supports a service-oriented business modeling paradigm.
- Contemporary SOA implements layers of abstraction.
- Contemporary SOA promotes loose coupling throughout the enterprise.
- Contemporary SOA promotes organizational agility.
- Contemporary SOA is a building block.
- Contemporary SOA is an evolution.

- Contemporary SOA is still maturing.
- Contemporary SOA is an achievable ideal.

Note the absence of traditional architectural qualities such as "secure," "transactional," "reliable," and so on. These have been grouped into the "Contemporary SOA increases quality of service" characteristic. Chapters 6 and 7 explain how the evolving landscape of Web services specifications addresses typical quality of service (QoS) requirements.

As we step through the following sections we elaborate on each of the characteristics in our list and discuss their overall meaning to SOA. In doing so, we also build a formal definition of contemporary SOA.

### 3.2.1 Contemporary SOA is at the core of the service-oriented computing platform

Before we get into the actual meaning behind contemporary SOA, let's first discuss how the term "SOA" has been tossed about within the IT industry. Many argue that the manner in which SOA is used to qualify products, designs, and technologies elevates this term beyond one that simply relates to architecture. SOA, some believe, has become synonymous with an entire new world application computing platform.

Past terms used to identify distinct application computing platforms were often suffixed with the word "architecture" when the architecture was actually being referenced. The terms "client-server" or "n-tier," for example, can be used to classify a tool, an administration infrastructure, or an application architecture.

With SOA, however, the actual acronym has become a multi-purpose buzzword used frequently when discussing an application computing platform consisting of Web services technology and service-orientation principles. Because the acronym already represents the word "architecture" we are unfortunately subjected to statements that can be confusing.

Perhaps the best way to view it is that if a product, design, or technology is prefixed with "SOA," it is something that was (directly or indirectly) created in support of an architecture based on service-orientation principles. Along those same lines, this book, though partially titled "Service-Oriented Architecture," goes well beyond architectural boundaries to explore the contemporary service-oriented platform.

Because we positioned contemporary SOA as building upon and extending the primitive SOA model, we already have a starting point for our definition:

> *Contemporary SOA represents an architecture that promotes service-orientation through the use of Web services.*

### 3.2.2 Contemporary SOA increases quality of service

There is a definite need to bring SOA to a point where it can implement enterprise-level functionality as safely and reliably as the more established distributed architectures already do.

This relates to common quality of service requirements, such as:

- The ability for tasks to be carried out in a secure manner, protecting the contents of a message, as well as access to individual services.
- Allowing tasks to be carried out reliably so that message delivery or notification of failed delivery can be guaranteed.
- Performance requirements to ensure that the overhead imposed by SOAP message and XML content processing does not inhibit the execution of a task.
- Transactional capabilities to protect the integrity of specific business tasks with a guarantee that should the task fail, exception logic is executed.

Contemporary SOA is striving to fill the QoS gaps of the primitive SOA model. Many of the concepts and specifications we discuss in *Part II—SOA and WS-* Extensions* provide features that directly address quality of service requirements. For lack of a better term, we'll refer to an SOA that fulfills specific quality of service requirements as "QoS-capable."

### 3.2.3 Contemporary SOA is fundamentally autonomous

The service-orientation principle of autonomy requires that individual services be as independent and self-contained as possible with respect to the control they maintain over their underlying logic. This is further realized through message-level autonomy where messages passed between services are sufficiently intelligence-heavy that they can control the manner in which they are processed by recipient services.

SOA builds upon and expands this principle by promoting the concept of autonomy throughout solution environments and the enterprise. Applications comprised of autonomous services, for example, can themselves be viewed as composite, self-reliant services that exercise their own self-governance within service-oriented integration environments.

Later we explain how by creating service abstraction layers, entire domains of solution logic can achieve control over their respective areas of governance. This establishes a level of autonomy that can cross solution boundaries.

### 3.2.4 Contemporary SOA is based on open standards

Perhaps the most significant characteristic of Web services is the fact that data exchange is governed by open standards. After a message is sent from one Web service to another it travels via a set of protocols that is globally standardized and accepted.

Further, the message itself is standardized, both in format and in how it represents its payload. The use of SOAP, WSDL, XML, and XML Schema allow for messages to be fully self-contained and support the underlying agreement that to communicate, services require nothing more than a knowledge of each other's service descriptions. The use of an open, standardized messaging model eliminates the need for underlying service logic to share type systems and supports the loosely coupled paradigm.

Contemporary SOAs fully leverage and reinforce this open, vendor-neutral communications framework (Figure 3.5). An SOA limits the role of proprietary technology to the implementation and hosting of the application logic encapsulated by a service. The opportunity for inter-service communication is therefore always an option.

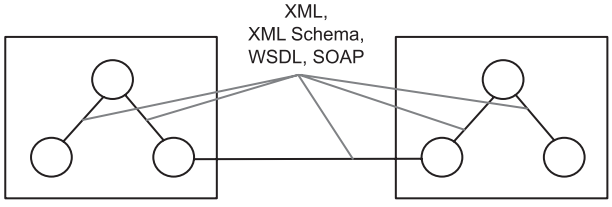



**Figure 3.5**
Standard open technologies are used within and outside of solution boundaries.

### 3.2.5 Contemporary SOA supports vendor diversity

The open communications framework explained in the previous section not only has significant implications for bridging much of the heterogeneity within (and between) corporations, but it also allows organizations to choose best-of-breed environments for specific applications.

For example, regardless of how proprietary a development environment is, as long as it supports the creation of standard Web services, it can be used to create a non-proprietary service interface layer, opening up interoperability opportunities with other, service-capable applications (Figure 3.6). This, incidentally, has changed the face of integration architectures, which now can encapsulate legacy logic through service adapters, and leverage middleware advancements based on Web services.

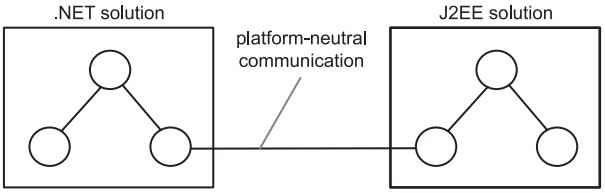



**Figure 3.6**
Disparate technology platforms do not prevent service-oriented solutions from interoperating.

Organizations can certainly continue building solutions with existing development tools and server products. In fact, it may make sense to do so, only to continue leveraging the skill sets of in-house resources. However, the choice to explore the offerings of new vendors is always there. This option is made possible by the open technology provided by the Web services framework and is made more attainable through the standardization and principles introduced by SOA.

### 3.2.6 Contemporary SOA promotes discovery

Even though the first generation of Web services standards included UDDI, few of the early implementations actually used service registries as part of their environments. This may have to do with the fact that not enough Web services were actually built to warrant a registry. However, another likely reason is that the concept of service discovery was simply not designed into the architecture. When utilized within traditional distributed architectures, Web services were more often employed to facilitate point-to-point solutions. Therefore, discovery was not a common concern.

SOA supports and encourages the advertisement and discovery of services throughout the enterprise and beyond. A serious SOA will likely rely on some form of service registry or directory to manage service descriptions (Figure 3.7).

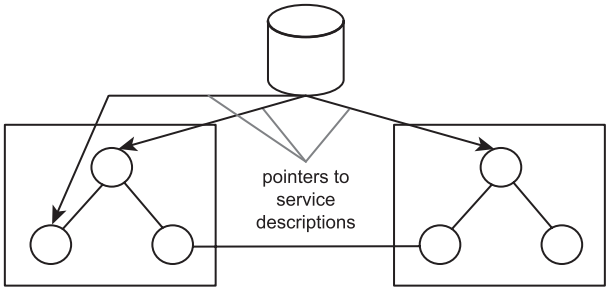



**Figure 3.7**
Registries enable a mechanism for the discovery of services.

### 3.2.7 Contemporary SOA fosters intrinsic interoperability

Further leveraging and supporting the required usage of open standards, a vendor diverse environment, and the availability of a discovery mechanism, is the concept of intrinsic interoperability. Regardless of whether an application actually has immediate integration requirements, design principles can be applied to outfit services with characteristics that naturally promote interoperability.

When building an SOA application from the ground up, services with intrinsic interoperability become potential integration endpoints (Figure 3.8). When properly standardized, this leads to service-oriented integration architectures wherein solutions themselves achieve a level of intrinsic interoperability. Fostering this characteristic can significantly alleviate the cost and effort of fulfilling future cross-application integration requirements.

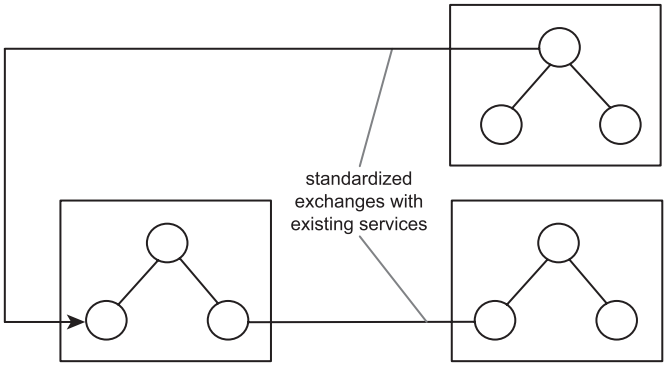



**Figure 3.8**
Intrinsically interoperable services enable unforeseen integration opportunities.

### 3.2.8 Contemporary SOA promotes federation

Establishing SOA within an enterprise does not necessarily require that you replace what you already have. One of the most attractive aspects of this architecture is its ability to introduce unity across previously non-federated environments. While Web services enable federation, SOA promotes this cause by establishing and standardizing the ability to encapsulate legacy and non-legacy application logic and by exposing it via a common, open, and standardized communications framework (also supported by an extensive adapter technology marketplace).

Obviously, the incorporation of SOA with previous platforms can lead to a variety of hybrid solutions. However, the key aspect is that the communication channels achieved by this form of service-oriented integration are all uniform and standardized (Figure 3.9).

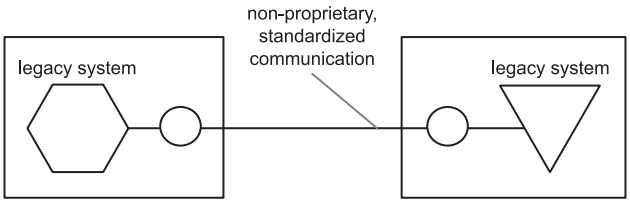



**Figure 3.9**
Services enable standardized federation of disparate legacy systems.

### 3.2.9 Contemporary SOA promotes architectural composability

Composability is a deep-rooted characteristic of SOA that can be realized on different levels. For example, by fostering the development and evolution of composable services, SOA supports the automation of flexible and highly adaptive business processes. As previously mentioned, services exist as independent units of logic. A business process can therefore be broken down into a series of services, each responsible for executing a portion of the process.

A broader example of composability is represented by the second-generation Web services framework that is evolving out of the release of the numerous WS-* specifications. The modular nature of these specifications allows an SOA to be composed of only the functional building blocks it requires.

What provides this flexibility is the fact that second-generation Web services specifications are being designed specifically to leverage the SOAP messaging model. Individual specifications consist of modular extensions that provide one or more specific features. As the offering of WS-* extensions supported by a given vendor platform grows, the flexibility to compose allows you to continue building solutions that only implement the features you actually need (Figure 3.10). In other words, the WS-* platform allows for the creation of streamlined and optimized service-oriented architectures, applications, services, and even messages.

With respect to our definition, let's represent this characteristic by describing the architecture as a whole as being composable. This represents both composable services, as well as the extensions that comprise individual SOA implementations.

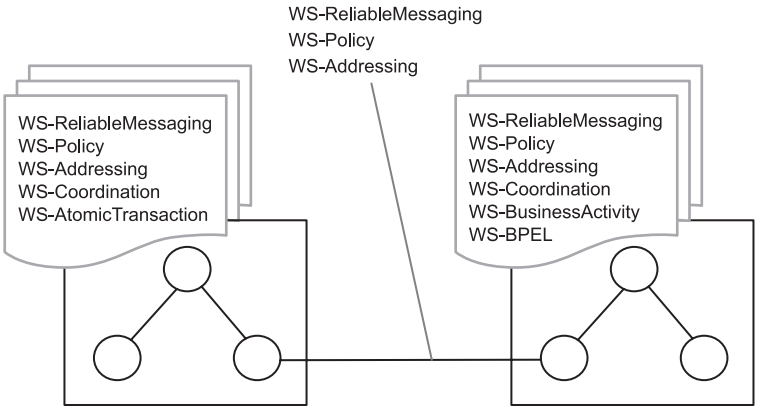



**Figure 3.10**
Different solutions can be composed of different extensions and can continue to interoperate as long as they support the common extensions required.

### 3.2.10 Contemporary SOA fosters inherent reusability

SOA establishes an environment that promotes reuse on many levels. For example, services designed according to service-orientation principles are encouraged to promote reuse, even if no immediate reuse requirements exist. Collections of services that form service compositions can themselves be reused by larger compositions.

The emphasis placed by SOA on the creation of services that are agnostic to both the business processes and the automation solutions that utilize them leads to an environment in which reuse is naturally realized as a side benefit to delivering services for a given project. Thus, inherent reuse can be fostered when building service-oriented solutions (Figure 3.11).

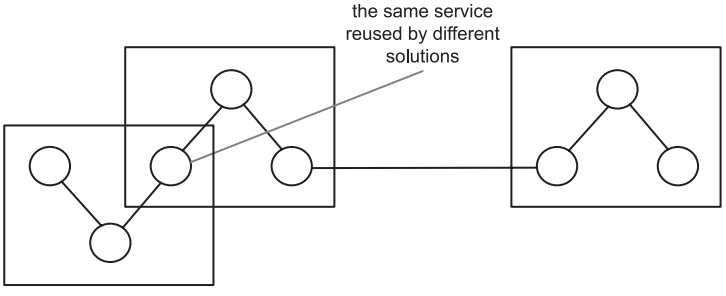



**Figure 3.11**
Inherent reuse accommodates unforeseen reuse opportunities.

### 3.2.11 Contemporary SOA emphasizes extensibility

When expressing encapsulated functionality through a service description, SOA encourages you to think beyond immediate, point-to-point communication requirements. When service logic is properly partitioned via an appropriate level of interface granularity, the scope of functionality offered by a service can sometimes be extended without breaking the established interface (Figure 3.12).

Extensibility is also a characteristic that is promoted throughout SOA as a whole. Extending entire solutions can be accomplished by adding services or by merging with other service-oriented applications (which also, effectively, "adds services"). Because the loosely coupled relationship fostered among all services minimizes inter-service dependencies, extending logic can be achieved with significantly less impact.

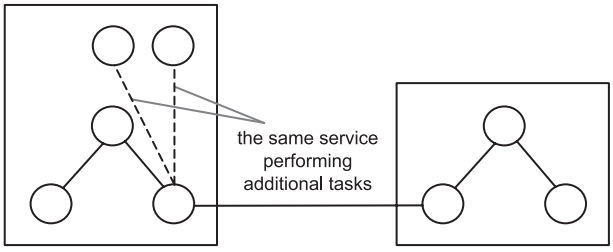



**Figure 3.12**
Extensible services can expand functionality with minimal impact.

Time to revisit our original definition to add a few adjectives that represent the characteristics we've covered.

> *Contemporary SOA represents an* ***open***, ***extensible***, ***federated***, ***composable*** *architecture that promotes service-orientation and is* ***comprised of autonomous***, ***QoS-capable***, ***vendor diverse***, ***interoperable***, ***discoverable***, *and* ***potentially reusable*** *services, implemented as Web services.*

### 3.2.12 Contemporary SOA supports a service-oriented business modeling paradigm

In our description of a primitive SOA, we briefly explored how business processes can be represented and expressed through services. Partitioning business logic into services that can then be composed has significant implications as to how business processes can be modeled (Figure 3.13). Analysts can leverage these features by incorporating an extent of service-orientation into business processes for implementation through SOAs.

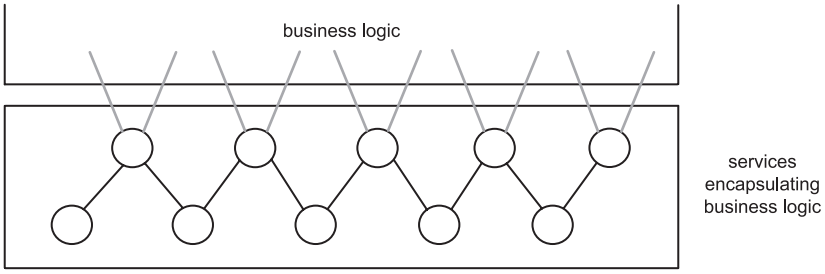



**Figure 3.13**
A collection (layer) of services encapsulating business process logic.

In other words, services can be designed to express business logic. BPM models, entity models, and other forms of business intelligence can be accurately represented through the coordinated composition of business-centric services. This is an area of SOA that is not yet widely accepted or understood. We therefore spend a significant portion of this book exploring the service-oriented business modeling paradigm.

### 3.2.13 Contemporary SOA implements layers of abstraction

One of the characteristics that tends to evolve naturally through the application of service-oriented design principles is that of abstraction. Typical SOAs can introduce layers of abstraction by positioning services as the sole access points to a variety of resources and processing logic.

When applied through proper design, abstraction can be targeted at business and application logic. For example, by establishing a layer of endpoints that represent entire solutions and technology platforms, all of the proprietary details associated with these environments disappear (Figure 3.14). The only remaining concern is the functionality offered via the service interfaces.

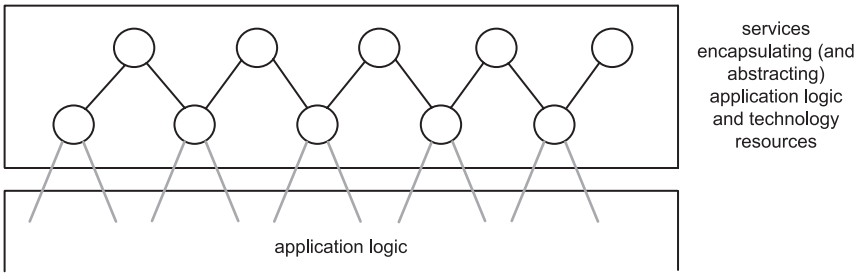



**Figure 3.14**
Application logic created with proprietary technology can be abstracted through a dedicated service layer.

It is the mutual abstraction of business and technology that supports the service-oriented business modeling paradigm we discussed and further establishes the loosely coupled enterprise model explained in the following section.

### 3.2.14 Contemporary SOA promotes loose coupling throughout the enterprise

As we've established, a core benefit to building a technical architecture with loosely coupled services is the resulting independence of service logic. Services only require an awareness of each other, allowing them to evolve independently.

Now, let's take a step back and look at the enterprise as a whole. Within an organization where service-orientation principles are applied to both business modeling and technical design, the concept of loose coupling is amplified.

By implementing standardized service abstraction layers, a loosely coupled relationship also can be achieved between the business and application technology domains of an enterprise (Figure 3.15). Each end only requires an awareness of the other, therefore allowing each domain to evolve more independently. The result is an environment that can better accommodate business and technology-related change—a quality known as organizational agility.

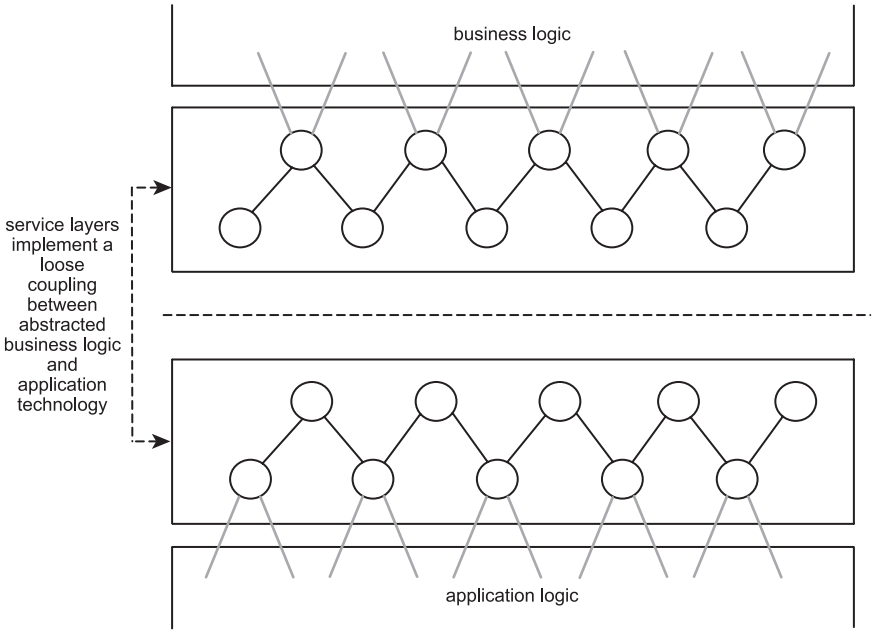



**Figure 3.15**
Through the implementation of service layers that abstract business and application logic, the loose coupling paradigm can be applied to the enterprise as a whole.

### 3.2.15 Contemporary SOA promotes organizational agility

Whether the result of an internal reorganization, a corporate merger, a change in an organization's business scope, or the replacement of an established technology platform, an organization's ability to accommodate change determines the efficiency with which it can respond to unplanned events.

Change in an organization's business logic can impact the application technology that automates it. Change in an organization's application technology infrastructure can impact the business logic automated by this technology. The more dependencies that exist between these two parts of an enterprise, the greater the extent to which change imposes disruption and expense.

By leveraging service business representation, service abstraction, and the loose coupling between business and application logic provided through the use of service layers, SOA offers the potential to increase organizational agility (Figure 3.16).

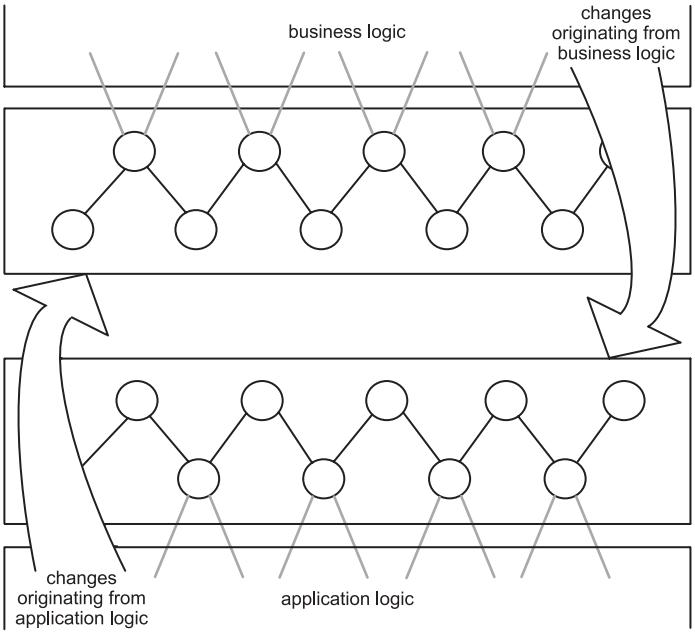



**Figure 3.16**
A loosely coupled relationship between business and application technology allows each end to more efficiently respond to changes in the other.

Other benefits realized through the standardization of SOA also contribute to minimizing dependencies and increasing overall responsiveness to change: notably, the intrinsic interoperability that can be built into services and the open communications framework established across integration architectures that enable interoperability between disparate platforms. Change imposed on any of these environments is more easily facilitated for the same reasons—a loosely coupled state between services representing either ends of the communication channel.

Organizational agility is perhaps the most significant benefit that can be realized with contemporary SOA.

### 3.2.16 Contemporary SOA is a building block

A service-oriented application architecture will likely be one of several within an organization committed to SOA as the standard architectural platform. Organizations standardizing on SOA work toward an ideal known as the service-oriented enterprise (SOE), where all business processes are composed of and exist as services, both logically and physically.

When viewed in the context of SOE, the functional boundary of an SOA represents a part of this future-state environment, either as a standalone unit of business automation or as a service encapsulating some or all of the business automation logic. In responding to business model-level changes, SOAs can be augmented to change the nature of their automation, or they can be pulled into service-oriented integration architectures that require the participation of multiple applications.

What this all boils down to is that an individual service-oriented application can, in its entirety, be represented by and modeled as a single service. As mentioned earlier, there are no limits to the scope of service encapsulation. An SOA consists of services within services within services, to the point that a solution based on SOA itself is one of many services within an SOE.

This past set of characteristics has further broadened our definition. Let's append the definition with the following:

> *SOA can establish an abstraction of business logic and technology that may introduce changes to business process modeling and technical architecture, resulting in a loose coupling between these models. These changes foster service-orientation in support of a service-oriented enterprise.*

### 3.2.17 Contemporary SOA is an evolution

SOA defines an architecture that is related to but still distinct from its predecessors. It differs from traditional client-server and distributed environments in that it is heavily influenced by the concepts and principles associated with service-orientation and Web services. It is similar to previous platforms in that it preserves the successful characteristics of its predecessors and builds upon them with distinct design patterns and a new technology set.

For example, SOA supports and promotes reuse, as well as the componentization and distribution of application logic. These and other established design principles that are commonplace in traditional distributed environments are still very much a part of SOA.

### 3.2.18 Contemporary SOA is still maturing

While the characteristics described so far are fundamental to contemporary SOA, this point is obviously more of a subjective statement of where SOA is at the moment. Even though SOA is being positioned as the next standard application computing platform, this transition is not yet complete. Despite the fact that Web services are being used to implement a great deal of application functionality, the support for a number of features necessary for enterprise-level computing is not yet fully available.

Standards organizations and major software vendors have produced many specifications to address a variety of supplementary extensions. Additionally, the next generation of development tools and application servers promises to support a great deal of these new technologies. When SOA platforms and tools reach an adequate level of maturity, the utilization of Web services can be extended to support the creation of enterprise SOA solutions, making the ideal of a service-oriented enterprise attainable.

If you needed to provide an accurate definition of SOA today, you would not be out of line to mention the state of its underlying technology. Considering the rate at which the IT industry as a whole is adopting and evolving the SOA platform, though, it should not be too long before an accurate definition will no longer require this statement.

### 3.2.19 Contemporary SOA is an achievable ideal

A standardized enterprise-wide adoption of SOA is a state to which many organizations would like to fast-forward. The reality is that the process of transitioning to this state demands an enormous amount of effort, discipline, and, depending on the size of the organization, a good amount of time. Every technical environment will undergo

changes during such a migration, and various parts of SOA will be phased in at different stages and to varying extents. This will likely result in countless hybrid architectures, consisting mostly of distributed environments that are part legacy and part service-oriented.

Further supporting this prediction is the evolving state of the technology set that is emerging to realize enterprise-level SOAs. As companies adopt SOA during this evolution, many will need to retrofit their environments (and their standards) to accommodate changes and innovations as SOA-related specifications, standards, and products continue to mature.

However, the majority of the contemporary SOA characteristics we just covered are attainable today. This book provides a series of tutorials and step-by-step process descriptions that explain how to manifest them.

### 3.2.20 Defining SOA

Now that we've finished covering characteristics, we can finalize our formal definition.

*Contemporary SOA represents an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services.*

*SOA can establish an abstraction of business logic and technology that may introduce changes to business process modeling and technical architecture, resulting in a loose coupling between these models.*

*SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise.*

*SOA is ideally standardized throughout an enterprise, but achieving this state requires a planned transition and the support of a still evolving technology set.*

Though accurate, this definition of contemporary SOA is quite detailed. For practical purposes, let's provide a supplementary definition that can be applied to both primitive and contemporary SOA.

*SOA is a form of technology architecture that adheres to the principles of service-orientation. When realized through the Web services technology platform, SOA establishes the potential to support and promote these principles throughout the business process and automation domains of an enterprise.*

### 3.2.21 Separating concrete characteristics

Looking back at the list of characteristics we just covered, we can actually split them into two groups—characteristics that represent concrete qualities that can be realized as real extensions of SOA and those that can be categorized as commentary or observations. Collectively, these characteristics were useful for achieving our formal definition. From here on, though, we are more interested in exploring the concrete characteristics only.

Let's therefore remove the following items from our original list:

- Contemporary SOA is at the core of the service-oriented computing platform.
- Contemporary SOA is a building block.
- Contemporary SOA is an evolution.
- Contemporary SOA is still maturing.
- Contemporary SOA is an achievable ideal.

By trimming these items, along with some superfluous wording, we end up with the following set of concrete characteristics.

Contemporary SOA is generally:

- based on open standards
- architecturally composable
- capable of improving QoS

Contemporary SOA supports, fosters, or promotes:

- vendor diversity
- intrinsic interoperability
- discoverability
- federation
- inherent reusability
- extensibility
- service-oriented business modeling
- layers of abstraction
- enterprise-wide loose coupling
- organizational agility

It is these characteristics that, when realized, provide tangible, measurable benefits.

NOTE

Though we qualify these as "concrete" here, it is this set of characteristics that we refer to when we use the term "contemporary SOA characteristics" from here on.

**SUMMARY OF KEY POINTS**

- We distinguish contemporary SOA with a number of common characteristics that build upon and extend the original qualities and principles established by primitive SOA.
- The realization of contemporary SOA characteristics is explored in detail throughout this book.

## 3.3 Common misperceptions about SOA

You'll often hear about the shift in mindset required to fully adopt SOA. The myths we tackle here only scratch the surface of the change that's in store for those firmly entrenched with traditional architectural views. Our focus in this section is to dispel the most common points of confusion about SOA and the use of the term "service-oriented."

### 3.3.1 "An application that uses Web services is service-oriented."

This comes down to how SOA is defined. As we've established, there is a distinction between SOA as an abstract model and SOA based on Web services and service-orientation. Depending on which abstract model you use, almost any form of distributed architecture can be classified as being service-oriented.

To realize the benefit potential of the mainstream variation of SOA we've been discussing, you need to standardize how Web services are positioned and designed, according to service-orientation principles.

So whether this statement is a misperception or not actually depends on your expectations. A traditional distributed architecture can be called service-oriented as long as the benefits associated with primitive and contemporary SOA are not expected.

### 3.3.2 "SOA is just a marketing term used to re-brand Web services."

Certainly the term "SOA" has been (appropriately and otherwise) used excessively for marketing purposes. It has become a high-profile buzzword, riding the wave of hype brought on by the rise of Web services. The fact that contemporary SOAs are being implemented using Web services has led to some skepticism around the validity of the term itself. Some believe that "SOA support" is just a re-labeling of "Web services support."

SOA is not an invention of the media or some marketing department. SOA is a legitimate and relatively established technical term. It represents a distinct architecture based on a set of distinct principles. It just happens that contemporary SOA also implies the use of a distinct set of technology used to realize fundamental SOA principles. The technology platform of choice is currently Web services and all that comes with it.

### 3.3.3 "SOA is just a marketing term used to re-brand distributed computing with Web services."

There are many people who believe this, which has led to the "false SOA" syndrome explained in Chapter 1. The reasons behind this myth are understandable. Much of the hype surrounding SOA has overshadowed its actual meaning. Additionally, many of the migration paths laid out by product vendors blend traditional distributed computing with Web services, accompanied by advertised "SOA support." The results certainly can be confusing. Further, the validity of some promoted SOA support is questionable at best.

However, SOA is its own entity. It consists of a set of design principles that are related to, but differ significantly from, the distributed computing platforms of the past. We compare SOA characteristics to distributed computing in detail in the following chapter.

### 3.3.4 "SOA simplifies distributed computing."

The principles behind SOA are relatively simple in nature. However, applying these principles in real life can prove to be a complex task. Though SOA offers significant benefit potential, this can only be realized by investing in thorough analysis and adherence to the design principles of service-orientation.

Typical SOA implementations require more up-front research than solutions created under previous platform paradigms. This is in part due to the broad Web services technology platform imposed by contemporary SOA.

The quality of simplicity surfaces later, once service-orientation is established and standardized within an IT environment. Then, when integration requirements emerge, when a sufficient number of composable services exist, and when service-orientation principles are well integrated into an organization, that's when SOA can simplify the delivery of automation requirements.

### 3.3.5 "An application with Web services that uses WS-* extensions is service-oriented."

While the entire second generation of Web services specifications are certainly driving SOA into the IT mainstream, simply making these extensions part of an architecture does not make it service-oriented. Regardless of the functionality with which Web services are outfitted, what makes them part of a service-oriented architecture is how the architecture itself is designed.

Having stated that, though, it is expected that most solutions that seriously employ the use of WS-* extensions will, in fact, be service-oriented. This is partially because the adoption rate of SOA is anticipated to roughly coincide with the availability of WS-* extension support in development and middleware products.

### 3.3.6 "If you understand Web services you won't have a problem building SOA."

A technical and conceptual knowledge of Web services is certainly helpful. However, as we established at the beginning of this chapter, fundamental service-orientation principles are pretty much technology agnostic. Service-orientation requires a change in how business and application logic are viewed, partitioned, and automated. It therefore also requires that Web services be utilized and designed according to specific principles.

Web services are easily incorporated into existing traditional distributed architectures. There they can be centrally positioned and assigned significant processing responsibilities, or they can be appended as peripheral application endpoints.

The manner in which Web services are utilized in SOA is significantly different. The emphasis placed on business logic encapsulation and the creation of service abstraction layers often will require a blend of technology and business analysis expertise. It is best to assume that realizing contemporary SOA requires a separate skill set that goes beyond a knowledge of Web services technology.

### 3.3.7 "Once you go SOA, everything becomes interoperable."

The media attention and marketing push behind SOA has likely contributed to this myth. Many assume that by virtue of building service-oriented solutions, their technical environments will naturally transform into a united, federated enterprise.

Though this ultimate goal is attainable, it requires investment, analysis, and, above all, a high degree of standardization. By leveraging the open Web services communications framework, service-oriented architectures (and service-oriented integration architectures) naturally abstract and hide all that is proprietary about a particular solution, its platform, and its technology.

This establishes a predictable communications medium for all applications exposed via a Web service. However, it does not automatically standardize the representation of the information that is exchanged via this medium. Therefore, as SOAs become more common, there will be good and not so good implementations. A quality SOA requires that individual services conform to common design standards for federation, interoperability, reusability, and other benefits to be fully realized.

**SUMMARY OF KEY POINTS**

- Much of the confusion surrounding the meaning of SOA is caused by how this term has been used by the media and in marketing literature.
- The most common misperceptions relate to the use of Web services within distributed Internet architectures being mistaken as contemporary SOA.
- Some of the more dangerous assumptions about SOA are that service-oriented solutions are simple by nature, easy to build, and automatically interoperable.

## 3.4 Common tangible benefits of SOA

So far we've discussed what constitutes an SOA. Much of this book expands on this topic by providing details about the inner workings of service-oriented solutions. Provided in this section is a list of the reasons why the IT community is going through the trouble of changing so much of its philosophy and technology in an effort to adopt SOA.

The benefits described in this section focus on tangible returns on investment, related primarily to:

- how SOA leads to improvements in automated solution construction
- how the proliferation of service-orientation ends up benefiting the enterprise as a whole

NOTE

SOA will benefit organizations in different ways, depending on their respective goals and the manner in which SOA and its supporting cast of products and technologies is applied. This list of common benefits is generalized and certainly not complete. It is merely an indication of the potential this architectural platform has to offer.

### 3.4.1 Improved integration (and intrinsic interoperability)

SOA can result in the creation of solutions that consist of inherently interoperable services. Utilizing solutions based on interoperable services is part of service-oriented integration (SOI) and results in a service-oriented integration architecture.

Because of the vendor-neutral communications framework established by Web services-driven SOAs, the potential is there for enterprises to implement highly standardized service descriptions and message structures. The net result is intrinsic interoperability, which turns a cross-application integration project into less of a custom development effort, and more of a modeling exercise.

The bottom line: The cost and effort of cross-application integration is significantly lowered when applications being integrated are SOA-compliant.

### 3.4.2 Inherent reuse

Service-orientation promotes the design of services that are inherently reusable. Designing services to support reuse from the get-go opens the door to increased opportunities for leveraging existing automation logic.

Building service-oriented solutions in such a manner that services fulfill immediate application-level requirements while still supporting a degree of reuse by future potential requestors establishes an environment wherein investments into existing systems can potentially be leveraged and re-leveraged as new solutions are built.

The bottom line: Building services to be inherently reusable results in a moderately increased development effort and requires the use of design standards. Subsequently leveraging reuse within services lowers the cost and effort of building service-oriented solutions.

### 3.4.3 Streamlined architectures and solutions

The concept of composition is another fundamental part of SOA. It is not, however, limited to the assembly of service collections into aggregate services. The WS-* platform is based in its entirety on the principle of composability.

As described in the *Common characteristics of contemporary SOA* section, this aspect of service-oriented architecture can lead to highly optimized automation environments, where only the technologies required actually become part of the architecture.

The bottom line: Realizing this benefit requires adherence to design standards that govern allowable extensions within each application environment. Benefits of streamlined solutions and architectures include the potential for reduced processing overhead and reduced skill-set requirements (because technical resources require only the knowledge of a given application, service, or service extension).

NOTE

The reduced performance requirements mentioned previously only refer to the fact that SOA extensions are composable and therefore allow each application-level architecture to contain extensions only relevant to its solution requirements. Message-based communication in SOAs can, in fact, increase performance requirements when compared to RPC-style communication within traditional distributed architectures. See the *Not understanding SOA performance requirements* section later in this chapter for more information.

### 3.4.4 Leveraging the legacy investment

The industry-wide acceptance of the Web services technology set has spawned a large adapter market, enabling many legacy environments to participate in service-oriented integration architectures. This allows IT departments to work toward a state of federation, where previously isolated environments now can interoperate without requiring the development of expensive and sometimes fragile point-to-point integration channels.

Though still riddled with risks relating mostly to how legacy back-ends must cope with increased usage volumes, the ability to use what you already have with service-oriented solutions that you are building now and in the future is extremely attractive.

The bottom line: The cost and effort of integrating legacy and contemporary solutions is lowered. The need for legacy systems to be replaced is potentially lessened.

### 3.4.5 Establishing standardized XML data representation

On its most fundamental level, SOA is built upon and driven by XML. As a result, an adoption of SOA leads to the opportunity to fully leverage the XML data representation platform. A standardized data representation format (once fully established) can reduce the underlying complexity of all affected application environments.

Examples include:

- XML documents and accompanying XML Schemas (packaged within SOAP messages) passed between applications or application components fully standardize format and typing of all data communicated. The result is a predictable and therefore easily extensible and adaptable communications network.
- XML's self-descriptive nature enhances the ability for data to be readily interpreted by architects, analysts, and developers. The result is the potential for data within messages to be more easily maintained, traced, and understood.
- The standardization level of data representation lays the groundwork for intrinsic interoperability. Specifically, by promoting the use of standardized vocabularies, the need to translate discrepancies between how respective applications have interpreted corporate data models is reduced.

Past efforts to standardize XML technologies have resulted in limited success, as XML was either incorporated in an ad-hoc manner or on an "as required" basis. These approaches severely inhibited the potential benefits XML could introduce to an organization. With contemporary SOA, establishing an XML data representation architecture becomes a necessity, providing organizations the opportunity to achieve a broad level of standardization.

The bottom line: The cost and effort of application development is reduced after a proliferation of standardized XML data representation is achieved.

| NOTE |
| --- |
| The last two benefits (legacy integration and XML data representation within SOA) are covered in *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*, the companion guide to this book. |

### 3.4.6 Focused investment on communications infrastructure

Because Web services establish a common communications framework, SOA can centralize inter-application and intra-application communication as part of standard IT infrastructure. This allows organizations to evolve enterprise-wide infrastructure by investing in a single technology set responsible for communication.

The bottom line: The cost of scaling communications infrastructure is reduced, as only one communications technology is required to support the federated part of the enterprise.

### 3.4.7 "Best-of-breed" alternatives

Some of the harshest criticisms laid against IT departments are related to the restrictions imposed by a given technology platform on its ability to fulfill the automation requirements of an organization's business areas. This can be due to the expense and effort required to realize the requested automation, or it may be the result of limitations inherent within the technology itself. Either way, IT departments are frequently required to push back and limit or even reject requests to alter or expand upon existing automation solutions.

SOA won't solve these problems entirely, but it is expected to increase empowerment of both business and IT communities. A key feature of service-oriented enterprise environments is the support of "best-of-breed" technology. Because SOA establishes a vendor-neutral communications framework, it frees IT departments from being chained to a single proprietary development and/or middleware platform. For any given piece of automation that can expose an adequate service interface, you now have a choice as to how you want to build the service that implements it.

The bottom line: The potential scope of business requirement fulfillment increases, as does the quality of business automation.

### 3.4.8 Organizational agility

Agility is a quality inherent in just about any aspect of the enterprise. A simple algorithm, a software component, a solution, a platform, a process—all of these parts contain a measure of agility related to how they are constructed, positioned, and leveraged. How building blocks such as these can be realized and maintained within existing financial and cultural constraints ultimately determines the agility of the organization as a whole.

Much of service-orientation is based on the assumption that what you build today will evolve over time. One of the primary benefits of a well-designed SOA is to protect organizations from the impact of this evolution. When accommodating change becomes the norm in distributed solution design, qualities such as reuse and interoperability become commonplace. The predictability of these qualities within the enterprise leads to a reliable level of organizational agility. However, all of this is only attainable through proper design and standardization.

Change can be disruptive, expensive, and potentially damaging to inflexible IT environments. Building automation solutions and supporting infrastructure with the anticipation of change seems to make a great deal of sense. A standardized technical environment comprised of loosely coupled, composable, and interoperable and potentially reusable services establishes a more adaptive automation environment that empowers IT departments to more easily adjust to change.

Further, by abstracting business logic and technology into specialized service layers, SOA can establish a loosely coupled relationship between these two enterprise domains. This allows each domain to evolve independently and adapt to changes imposed by the other, as required. Regardless of what parts of service-oriented environments are leveraged, the increased agility with which IT can respond to business process or technology-related changes is significant.

The bottom line: The cost and effort to respond and adapt to business or technology-related change is reduced.

**SUMMARY OF KEY POINTS**

- When assessing the return on investment for an SOA there are several concrete benefits that can be taken into account.
- However, many of the benefits promised by SOA do not manifest themselves until the use of service-orientation principles becomes established within an enterprise. As a result, there are few short-term benefits.

## 3.5 Common pitfalls of adopting SOA

Having just covered the benefit potential of contemporary SOA, it's time for a reality check. As with any application design or architecture, quality can vary. SOA is no exception. In fact, considering the extent to which organizations need to shift technology and mindset to fully adopt SOA, it is actually probable that some will inadvertently build

bad service-oriented architectures. Following are descriptions of some of the more common mistakes.

### 3.5.1 Building service-oriented architectures like traditional distributed architectures

Probably the number one obstacle organizations face in achieving SOA is building traditional distributed architectures under the pretense that they are building contemporary SOA. This is often the result of an acceptance of one or more of the misperceptions listed earlier in this chapter.

The danger with this scenario is that an organization can go quite far in terms of integrating the Web services technology set before realizing that they've been heading down the wrong path.

Examples of some of the problems this can introduce include:

- Proliferation of RPC-style service descriptions (leading to increased volumes of fine-grained message exchanges).
- Inhibiting the adoption of features provided by WS-* specifications.
- Improper partitioning of functional boundaries within services.
- Creation of non-composable (or semi-composable) services.
- Further entrenchment of synchronous communication patterns.
- Creation of hybrid or non-standardized services.

Understanding the fundamental differences between SOA and previous architectures is the key to avoiding this situation. (Part of Chapter 4 is dedicated to discussing this topic.)

### 3.5.2 Not standardizing SOA

In larger organizations where various IT projects occur concurrently, the need for custom standards is paramount. If different development projects result in the creation of differently designed applications, future integration efforts will be expensive and potentially fragile. This is a lesson many IT departments have already learned through past legacy nightmares.

The ability for SOA to achieve federation across disparate environments has been well promoted. The potential exists. However, it does not happen by simply purchasing the latest upgrades to a vendor's development tools and server software. SOA, like any

other architecture, requires the creation and enforcement of design standards for its benefits to be truly realized. (See the *"Once you go SOA, everything becomes interoperable"* myth described earlier in this chapter.)

For example, if one project builds a service-oriented solution in isolation from others, key aspects of its solution will not be in alignment with the neighboring applications it may be required to interoperate with one day.

This can lead to many problems, including:

- Incompatible data representation that results in disparate schemas representing the same types of information.
- Service descriptions with irregular interface characteristics and semantics.
- Support for different extensions or extensions being implemented in different ways.

SOA promotes a development environment that abstracts back-end processing so that it can execute and evolve independently within each application. However, standardization is still required to ensure consistency in design and interaction of services that encapsulate this back-end logic. Design standards, such as the "WSDL first" approach explored throughout Parts IV and V of this book, are required to realize many of the key benefits provided by SOA.

### 3.5.3 Not creating a transition plan

The chances of a successful migration will be severely diminished without the use of a comprehensive transition plan. Because the extent to which service endpoints are positioned within an enterprise can lead to a redefinition of an IT environment's infrastructure, the repercussions of a poorly executed migration can be significant.

Transition plans allow you to coordinate a controlled phasing in of service-orientation and SOA characteristics so that the migration can be planned on a technological, architectural, and organizational level.

Examples of typical areas covered by a transition plan include:

- An impact analysis that predicts the extent of change SOA will impose on existing resources, processes, custom standards, and technology.
- The definition of transition architectures, where those environments identified as candidates for a migration to SOA evolve through a series of planned hybrid stages.

- A speculative analysis that takes into account the future growth of Web services and supporting technologies.
- Detailed design changes to centralized logic (such as a new security model).

Creating a transition plan avoids the many problems associated with an ad-hoc adoption of SOA. Each plan, though, will be unique to an organization's requirements, constraints, and goals.

### 3.5.4 Not starting with an XML foundation architecture

In the world of contemporary SOA, everything begins with Web services. That statement has become a mantra of sorts within some organizations, but it is not entirely true. In the world of contemporary SOA, everything, in fact, begins with XML. It is the standard from which multiple supplementary standards have evolved to form a de facto data representation architecture. It is this core set of standards that has fueled the creation of the many Web services specifications that are now driving SOA.

So much attention is given to how data is transported between services that the manner in which this same data is structured and validated behind service lines is often neglected. This oversight can lead to an improper implementation of a persistent XML data representation layer within SOAs. The results can severely affect the quality of data processing. For example, the same data may be unnecessarily validated multiple times, or redundant data processing can inadvertently be performed before and after a service transmits or receives a message.

Standardizing the manner in which core XML technologies are used to represent, validate, and process corporate data as it travels throughout application environments (both within and between services) lays the groundwork for a robust, optimized, and interoperable SOA.

### 3.5.5 Not understanding SOA performance requirements

When starting out small, it is easy to build service-oriented solutions that function and respond as expected. As the scope increases and more functionality is added, the volume of message-based communication predictably grows. This is when unprepared environments can begin experiencing significant processing latency.

Because contemporary SOA introduces layers of data processing, it is subject to the associated performance overhead imposed by these layers. Contemporary SOA's reliance on Web services deepens its dependence on XML data representation, which, in

turn, can magnify XML processing-related performance challenges. For example, Web services security measures, such as encryption and digital signing, add new layers of processing to both the senders and recipients of messages.

Critical to building a successful service-oriented solution is understanding the performance requirements of your solution and the performance limitations of your infrastructure ahead of time.

This means:

- Testing the message processing capabilities of your environments prior to investing in Web services.
- Stress-testing the vendor supplied processors (for XML, XSLT, SOAP, etc.) you are planning to use.
- Exploring alternative processors, accelerators, or other types of supporting technology, if necessary. For example, the XML-binary Optimized Packaging (XOP) and SOAP Message Transmission Optimization Mechanism (MTOM) specifications developed by the W3C. (For more information, visit `www.w3c.org`.)

Performance is also one of the reasons coarse-grained service interfaces and asynchronous messaging are emphasized when building Web services. These and other design measures can be implemented to avoid potential processing bottlenecks.

### 3.5.6 Not understanding Web services security

The extent to which Web services technology grows within a given environment is typically related to the comfort level developers and architects have with the overall technology framework. Once it does expand it is easy to simply continue building on simplistic message exchanges, which usually rely on Secure Sockets Layer (SSL) encryption to implement a familiar measure of security.

While SSL can address many immediate security concerns, it is not the technology of choice for SOA. When services begin to take on greater amounts of processing responsibility, the need for message-level security begins to arise. The WS-Security framework establishes an accepted security model supported by a family of specifications that end up infiltrating service-oriented application and enterprise architectures on many levels.

One of the more significant design issues you may face when WS-Security hits your world is the potential introduction of centralized security. With this approach, the architecture abstracts a large portion of security logic and rules into a separate, central layer that is then relied upon by service-oriented applications.

Even if your vendor platform does not yet provide adequate support for WS-Security, and even if your current SSL-based implementation is meeting immediate requirements, it is also advisable to pay close attention to the changes that are ahead. Proceeding without taking WS-Security into account will inevitably lead to expensive retrofitting and redevelopment. This impact is amplified if you decide to implement a centralized security model, which would essentially become an extension of IT infrastructure. Acquiring a sound knowledge of the framework now will allow you to adjust your current architecture and application designs to better accommodate upcoming changes.

### 3.5.7 Not keeping in touch with product platforms and standards development

IT professionals used to working within the confines of a single development platform have become accustomed to focusing on industry trends as they apply to the product set they are currently working with. For example, .NET developers are generally not too concerned with what's happening in the Java world, and vice versa.

A transition to SOA opens up the arena of products and platforms that IT departments can choose from to build and/or host custom-developed application logic. While the tendency will be there to continue with what you know best, the option to look elsewhere is ever-present. As explained earlier, this is the result of establishing a vendor-neutral communications framework that allows solutions based on disparate technologies to become fully interoperable.

Another factor that can (and should) weigh in when comparing products is how product vendors relate to the WS-* specification development process that is currently underway. As different vendor alliances continue to produce competing extensions, how your vendors position themselves amidst this landscape will become increasingly important, especially once you begin to identify the extensions required to implement and execute key parts of your solutions' application logic.

In the meantime, specific aspects to look out for include:

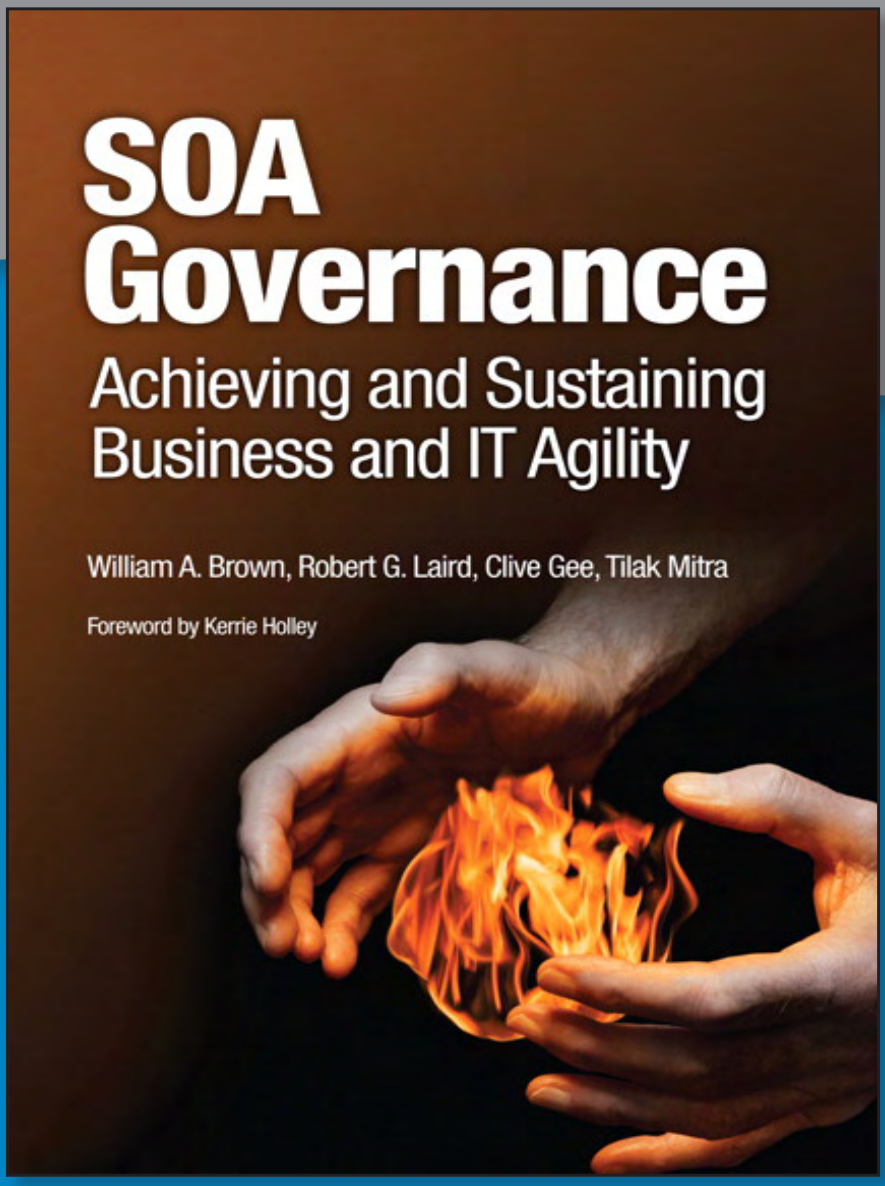
- which specifications vendors choose to support
- the involvement vendors demonstrate with the standards development process itself

- which other organizations vendors choose to ally themselves with (for a given standard)
- roadmaps published by vendors explaining how their product or platform will support upcoming specifications and standards

Chapter 4 provides an overview of the standards development process, including descriptions of the primary standards organizations related to SOA.

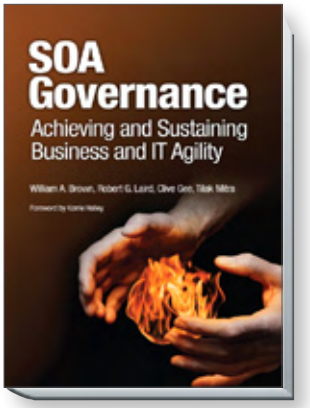
### SUMMARY OF KEY POINTS

- Many of the pitfalls relate to a limited understanding of what SOA is and what is required to fully incorporate and standardize service-orientation.
- A transition plan is the best weapon against the obstacles that tend to face organizations when migrating toward SOA.
- Staying in touch with commercial and standards-related developments is an important part of keeping an existing SOA aligned with future developments.



William A. Brown
Robert G. Laird
Clive Gee
Tilak Mitra

BUY ME




# SOA Governance

## Achieving and Sustaining Business and IT Agility

### Address the #1 Success Factor in SOA Implementations: Effective, Business-Driven Governance

Inadequate governance might be the most widespread root cause of SOA failure. In SOA Governance, a team of IBM's leading SOA governance experts share hard-won best practices for governing IT in any service-oriented environment.

The authors begin by introducing a comprehensive SOA governance model that has worked in the field. They define what must be governed, identify key stakeholders, and review the relationship of SOA governance to existing governance bodies as well as governance frameworks like COBIT. Next, they walk you through SOA governance assessment and planning, identifying and fixing gaps, setting goals and objectives, and establishing workable roadmaps and governance deliverables. Finally, the authors detail the build-out of the SOA governance model with a case study.

The authors illuminate the unique issues associated with applying IT governance to a services model, including the challenges of compliance auditing when service behavior is inherently unpredictable. They also show why services governance requires a more organizational, business-centric focus than "conventional" IT governance.

### About the Authors

WILLIAM A. BROWN is a Master and Senior Certiied Executive Architect with the IBM Global Business Services Enterprise Architecture and Technology Center of Excellence and the SOA Center of Excellence. He is the SOA Governance and Management Method (SGMM) global lead and the author of many of IBM's SOA governance methods and assets.

ROBERT G. LAIRD is an SOA Architect in the IBM Software SOA Advanced Technology Group and is responsible for supporting and leading SOA governance, SOA Architecture, and Telco Architecture engagements worldwide IBM customers. Bob has 30 years of industry and consulting experience.

CLIVE GEE Ph D, an Executive Consultant in the IBM Software SOA Advanced Technology Group, is one of IBM's most experienced international SOA governance practitioners, with 30 years of IT experience.

TILAK MITRA is a Senior Certiied Executive IT Architect with the IBM Global Business Services Enterprise Application Development group working closely with the SOA Center of Excellence. He specializes in executing on end-to-end SOA-based enterprise architectures along with its design and development.

AVAILABLE
- BOOK: 9780137147465
- SAFARI ONLINE
- EBOOK: 0137002920
- KINDLE: COMING SOON

COVERAGE INCLUDES

- Understanding the problems SOA governance needs to solve
- Establishing and governing service production lines that automate SOA development activities
- Identifying reusable elements of your existing IT governance model and prioritizing improvements
- Establishing SOA authority chains, roles, responsibilities, policies, standards, mechanisms, procedures, and metrics
- Implementing service versioning and granularity
- Refining SOA governance frameworks to maintain their vitality as business and IT strategies change

IBM Press™

ibmpressbooks.com

# 8

# SOA Governance Case Study

*"Far better is it to dare mighty things, to win glorious triumphs, even though checkered by failure… than to rank with those poor spirits who neither enjoy nor suffer much, because they live in a gray twilight that knows not victory nor defeat."*

**—Theodore Roosevelt**

We are taking the point of view for this book on SOA governance that a case study approach will help to drive home the points and help you understand and apply the practical aspects of planning and building a governance capability.

The case study uses some of the salient aspects of SOA governance from the book to provide an example of the application of the governance needs of a fictional company. This company, Ideation Communications, does not exist, and any resemblance to an existing company is purely coincidental.

# Introduction: Case Study Background

***"Walking is a man's best medicine."***

**—Hippocrates**

Ideation Communications started as local, independent communications service provider in the rural areas of the Midwestern portion of the United States, offering Plain Old Telephone Service (POTS) to satisfy the communication needs of a largely agrarian customer base separated by long distances.

Despite its business being initially rural, Ideation has always placed an emphasis on implementing technological innovation as quickly as it makes sense, for the benefit of its customers and workers. Ideation was one of the first communications providers to offer 100% dial service with no manual operators needed. Ideation employed microwave towers in the 1970s in its territory and leased the bandwidth to truckers and local companies. This soon grew into a national footprint that provided a long-distance telephony service that grew profits explosively. Data capabilities with ATM (Asynchronous Transfer Mode) and Frame Relay were added due to corporate demand. Ideation was one of the first companies to see the value of the Internet and chose to purchase a company (Inet) that had created an Internet network capability that both commercial and residential customers use.

Cell (mobile) phone was seen as a strategic differentiator, and Ideation built a strong presence of cell towers and service in its territory and has been trying to expand into a more national presence. There is a repeated emphasis on "the customer comes first," and the company strives to give customers the services that they want at a fair price, while at the same time providing a quality customer service experience at all of their stakeholder touch points.

On the employee side, Ideation has had another point of emphasis on "safety first." For example, in the 1950s, all field personnel involved in truck rolls were equipped with nylon body belts and climber straps when those were first available. Workers have been treated with respect, and their ideas are continuously solicited by management for getting the job done better, faster, and cheaper. As a result of the new markets that

Ideation has entered over time, there have continuously been new opportunities for personnel, and this, combined with the progressive treatment by management, has led to a high degree of worker loyalty and commitment.

Ideation is currently experiencing tough times, though. Long distance, a cash cow for several decades, has now been commoditized as a result of deregulation and usage of Voice over IP (VoIP) by many of its former and even current subscribers. Revenue has continued to decrease in this area by 10% a year, and shows no signs of stopping despite repeated marketing campaigns by Ideation's customer service centers. Its ATM and Frame Relay revenue and customers have also atrophied in favor of Internet services such as virtual private network (VPN). Cell phone service has grown about 15% per year, but is now coming under attack from bigger and better capitalized national providers.

Increasingly, management attention has been drawn to the many different systems and operations groups and the various networks that Ideation maintains, as something that is impacting their margins (see Figure 8.1).

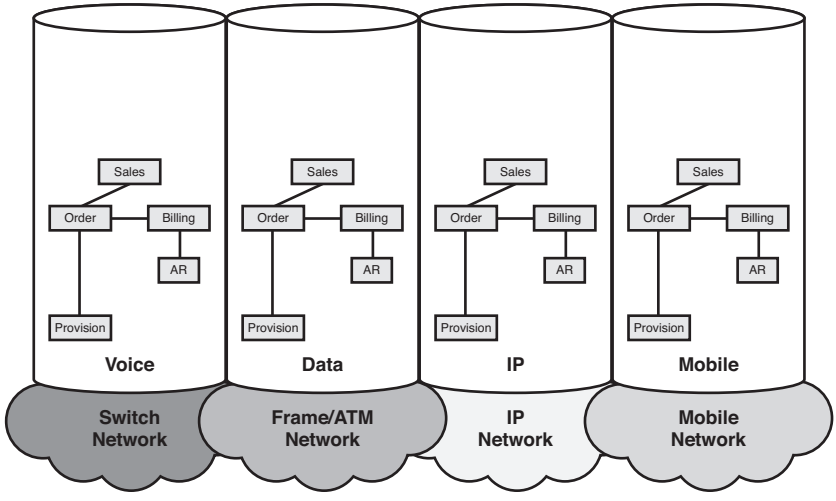



**Figure 8.1** Ideation Systems and Networks

There are many different ordering systems, provisioning systems, billing systems, and network management systems. Each of these systems has its own support structure, subject matter experts, business analysts,

management structure, and development cycles. Because the business has required products that involve bundles of existing network capabilities, the amount of time to get a new release out has continually increased with missed schedules, project de-scope, budget over runs, or all three.

A year ago, at the direction of the board of directors, Ideation assembled its best and brightest and some industry consultants to analyze the corporate and industry trends and the current situation for Ideation and to come up with the business goals and objectives that would enable it to reverse the negative trends. Some of their conclusions on the current state of Ideation included the following:

- We built up rapidly because of market opportunities and never took the time to integrate our business well. As a result, we are characterized by many separate lines of business (LoB).
- Our cost structure is too high. We have much duplication of services across our various LoB, and this is hampering our ability to significantly reduce costs and increase margins.
- Our ability to offer bundles is constrained by the silos that have been built up in our IT infrastructure. Integrating any of our products requires a huge IT effort with a minimum time frame of two years. This is constraining Ideation's ability to react to market opportunities.
- Our ability to gather information about our customers and to exploit our strategic relationship with them to sell new products and services is severely constrained. Customer data is distributed in various LoB databases, each having a different customer ID and name. Ideation is losing out on market opportunities against our current customer base as a result.
- Ideation is too small to survive as an independent entity. It is highly likely that we will be bought in the next three years. Our value will be enhanced to the degree that we are able to integrate our LOB and operations, decrease costs, improve margins, and improve our business agility. We have 12 months to get this going before we will be constrained by the investment environment.

In consultation with the board, Ideation Communications created the following business goals for the next year:

- Be able to create new product bundles in six month's time measured from product marketing ideation to product offering in the marketplace.
- Decrease the operating run rate by 10%.
- Create upsell opportunities with our current customers that result in an increased average revenue per user (ARPU) by 5%.
- Create a common services organization and merge duplicative corporate capabilities into this single organization. This will include finance, HR, and data center support.
- Create and implement a business agility capability that includes a governance organization and linkage to the business so that they can work together. For example, groups such as enterprise architecture, the business, and IT must work together to bring quality to Ideation's business and to achieve the goals Ideation has outlined.

The path ahead is a difficult one for Ideation. Although they still have a positive corporate culture, Ideation experienced its first layoff this year, and the staff is wary of any changes. Rumors about a merger with an acquiring company have been floating around for some time, further inflaming the angst of the staff.

You have been selected by the CIO to lead the governance organization and are working with a highly intelligent and motivated team that has been charged with the task of creating business agility. The first order of business is to do a governance planning assessment.

## SOA Governance Assessment and Planning

> ***"Let the young know they will never find a more interesting, more instructive book than the patient himself."***
>
> **—Giorgio Baglivi**

Now that the CIO has assigned you the task of implementing the SOA governance function at Ideation, your first step is to understand all the governance items that you must address. Although you know that you won't be able to attack all of them at once, nevertheless you want to get the big picture first.

## SOA Planning Assessment

The first order of business is to use the SOA governance planning assessment that was discussed in Chapter 2, "SOA Goverance Assessment and Planning," to understand Ideation's current state from an SOA governance and SOA program point of view.

You need to assess not only where the organization is today, but also the areas of greatest program and governance need. This enables you to establish priorities for the governance transition plan to be created. This not only gives you a structured planning tool, but it also gives you metrics to measure the success of the transition plan at periodic milestones. It also helps you by giving some standard metrics for the necessary discussions that you need to periodically have with the SOA program and governance executive leadership to communicate progress and accept feedback on areas to be focused on.

Using the SOA governance planning assessment, you will interview various stakeholders from both business and IT. This will consist of some 1:1 sessions so that you can get frank and open comments and in some team sessions where you will get information from the right people in a particular area. Topics will include the SOA governance assessment areas domains of Plan & Organize, Program Management Controls, Service Development Lifecycle, and Service Operations.

At Ideation, the assessment information you are getting is eye opening even though you thought you knew the organization quite well. The development process in the various organizations is haphazard and not well structured. One highly respected subject matter expert (SME) says to you, "The inmates are running the asylum here. I'm really not sure how we get anything done. It's only because we have heroes who step in at the last second and save the day that we get any releases out. Even then, we've cut so much of the scope that I wonder why we bother at all." Another manager says, "Of course we do everything right, but all of the other groups are idiots. How they keep their jobs, I'll never know." A project manager says to you, "There is no consequence management. The same people screw up, and everyone just shrugs. I'm sick of spending so many hours here to make things work when management doesn't seem to know or care."

Whew! You have to take a step back and think after the interview sessions. The reactions were passionate, and the people involved kept looking at you with a mixture of hope and cynicism. "Nothing around here ever really changes, will it?" one tester questions you. "When the going gets

tough, the execs around here will always compromise doing something meaningful and with quality for the sake of looking good to their bosses and meeting their dates." You patiently explain that that's why governance is being assessed and that their leadership, ideas, and passion are needed now more than ever.

Although there are many issues that governance should eventually address at Ideation, a few key ones stand out:

- The lack of a consistent process, repeatability, best practices, and quality controls are obvious. When some of those things have been tried in the past, they usually end up sacrificed upon the altar of project dates. Problems that should have been caught and resolved during business requirement, solution architecture, and design find their way to testing, where it is expensive and time-consuming to fix. Something that must be redesigned, recoded, or reassembled and retested is a long process.
- The company has been thinking about ideas around creating business agility, particularly across LoB. There's a strategy group somewhere in corporate that is alleged to have done some thinking about this, but some quick checking determines that the strategy group is an "ivory tower" organization that is not steeped in the realities of the business. Apparently, no group is really charged with the responsibility of figuring out what business services make sense, and consequently, each project team must figure it out on their own or just write something that works for their LoB. Chances of cross-organizational cooperation and success are low without planning and real thinking in this area. Also, there are some common IT services, but it is unclear what they are and who owns or maintains them. You know that there need to be some ground rules around service ownership and funding, especially when they affect cross-organizational projects that must come together as a program.
- It is not clear how the cross-department organizations should interact with each other. Who makes decisions when the various LoB disagree with each other and the goal is to create a capability that is cross organization? How does the work get managed across groups? How are common standards and principles set and promulgated across the organizational? How are reference architectures created and accepted and architectural decisions made? How does IT interact with the business? All of these must be addressed by governance, and the rules of engagement must be specified.

- Service certification has been a disaster. The rules of engagement are unknown, and operations continually complain that they are not consulted until it is too late and they have to fix things in production. Most of second- and third- level support are burned out after being called in continually on nights and weekends and several have quit.
- There is no vitality in any of the processes that do exist. *"Process"* is probably too strong a term for what actually happens; organized chaos is probably closer to reality. "If you don't measure it, you can't improve it," one program manager says to you, and you wholeheartedly agree.

You present the results of the survey to the Ideation leadership team from the SOA planning assessment, as noted in Table 8.1 with the values corresponding to the maturity levels noted in Chapter 2. It is agreed to go forward with the steps necessary to define the first phase of an SOA governance transition plan.

**Table 8.1** SOA Governance Planning Assessment Results for Ideation

| SOA Governance Planning Assessment | Ideation Rating | Desired Ideation Rating |
|---|---|---|
| *Plan and Organize* | | |
| Services Transformation Planning | 1.67 | 4.00 |
| Information Transformation Planning | 1.00 | 3.83 |
| Technology Transformation Planning | 2.12 | 4.17 |
| Service Processes, Organizations, Roles & Responsibilities | 1.00 | 4.00 |
| Manage the Service Investment | 0.50 | 4.50 |
| Business Vision & IT Alignment | 1.75 | 3.50 |
| Service Portfolio Management | 1.83 | 4.33 |
| Service Ownership & Funding | 2.00 | 4.00 |
| Service Governance Vitality | 0.00 | 3.00 |
| Service Communication Planning | 2.00 | 4.00 |
| Service Education & Training | 1.17 | 4.00 |
| **Plan and Organize Average** | 1.37 | 3.94 |
| *Program Management Controls* | | |
| Enterprise Program Management | 1.50 | 3.50 |
| Change Management | 1.00 | 3.00 |
| Procurement of Resources | 1.00 | 3.00 |

| SOA Governance Planning Assessment | Ideation Rating | Desired Ideation Rating |
|---|---|---|
| Vendor Management | 1.50 | 3.00 |
| Identify and Allocate Costs | 0.50 | 2.50 |
| Monitor Business Benefits of SOA | 0.50 | 3.00 |
| **Program Management Controls Average** | 1.00 | 3.00 |
| *Service Development* | | |
| Service Development Lifecycle Controls | 1.00 | 4.00 |
| Manage Business Requirements | 1.00 | 4.00 |
| Service Identification | 0.50 | 3.00 |
| Service Specification | 1.00 | 3.00 |
| Service Realization | 1.50 | 3.50 |
| Service Certification | 0.33 | 3.50 |
| **Service Development Average** | 0.89 | 3.50 |
| *Service Operations* | | |
| Service Execution Monitoring | 2.00 | 3.00 |
| Service Operational Vitality | 1.67 | 3.00 |
| Service Support | 1.33 | 3.00 |
| **Service Operations Average** | 1.67 | 3.00 |
| **SOA Governance Planning Average** | 1.21 | 3.51 |

# Building the Service Factory

> ***"In all planning you make a list and you set priorities"***
>
> **—Alan Lakein**

Now that you have successfully completed the SOA governance planning assessment, you are keen to consider the various SOA governance capabilities and figure out where you should be focusing your limited resources.

## Service Transformation Planning

You have agreed with the stakeholders about the immediate areas that need to be addressed in phase one of the SOA governance definition and enablement. A heat map has been created to show the results (see Figure 8.2).

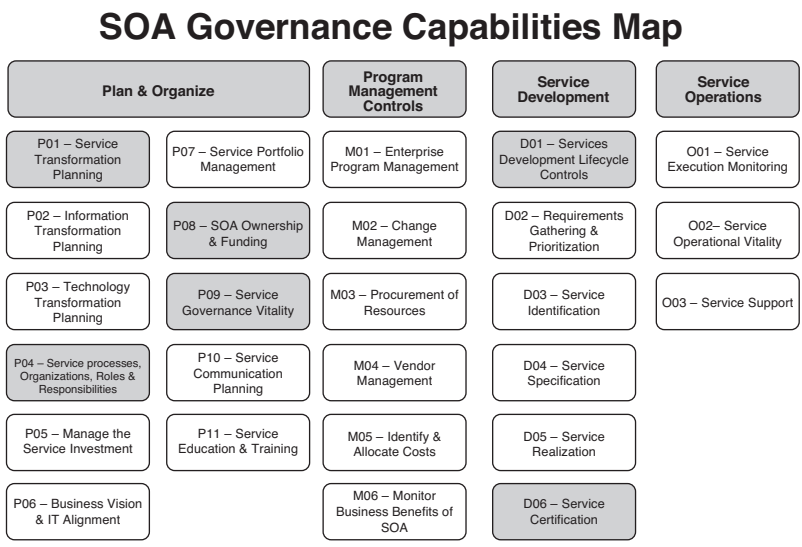



**Figure 8.2** SOA Governance Capabilities Heat Map for Ideation

Service transformation planning is an area that many of the most important stakeholders are concerned about. Business agility is one of the main drivers for SOA adoption, and creation of a service transformation plan is important in leading the organization to that goal. SOA governance must ensure that the right resources are engaged in this task.

The Enterprise Architecture (EA) group at Ideation has been studying this and has decided to use a standard industry business function map that can be used to map the enterprise's current business processes. They want to use this map to identify the current business groups and map their business responsibilities to standard business functions to understand how business opportunities can be grouped together. The EA group will use a gap and overlap analysis to identify areas of strength and areas of opportunity for business agility.

Ideation chose to use the TeleManagement Forum (TMF) Next Generation Operating System Support and its standard business process framework called enhanced Telecommunications Operations Map (eTOM; www.tmforum.org/browse.aspx? catID=1648). Figure 8.3 shows some of the standard Level 2 business functions that a typical telecommunications firm would be using, as per the industry TeleManagement Forum.

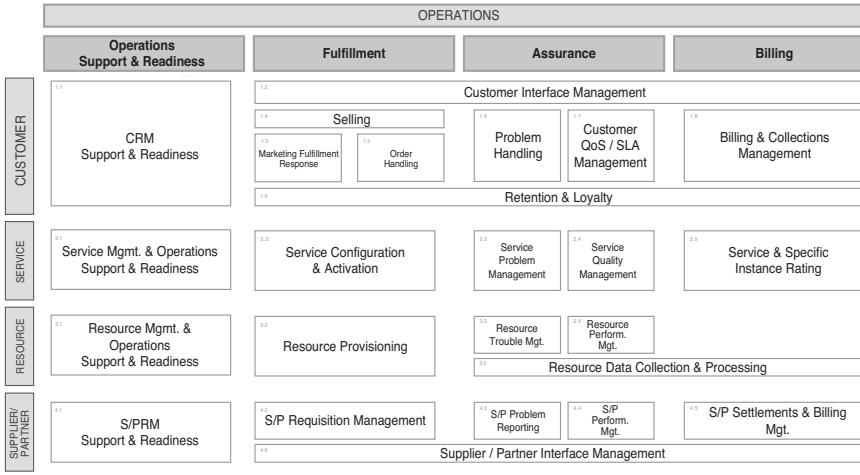



**Figure 8.3** Use an Industry Functional Component Model to Guide the Business Agility Discussion

For Ideation, the enterprise architecture group uses their industry function map in interviews with the business units and identifies that four separate business groups perform the business function of "2.5 Instance & Service-Specific Rating." In some cases, different pricing is available on the same service from a different sales group within the company. Consequently, it is a business goal to create a single rating function for all Ideation Communication products, and an SOA business service has been created with the name Create Rating. This service then consists of a set of subservices—Mediate Usage Records, Rate Usage Records, and Analyze Usage Records—as per the eTOM Level 3 business functional map.

In this manner, an industry business process map drives both a business discussion that results in measurable benefit to the business and the planning of the SOA business services creation strategy. In addition, the business agility discussion is a good time to establish key performance indicators (KPIs). In the example on rating, an obvious KPI created might be "single source of pricing for all products by the end of the year." A more business-focused KPI in this case might be "DSL product margin increase 2% due to elimination of pricing mistakes."

As a result of this analysis, the EA group creates a business architecture that identifies service domains, business processes, and an optimal set of business services that are prioritized as to usefulness to Ideation.

The SOA governance function ensures that this standard is used in areas such as investment decision making and opportunities for service creation, to inform and expedite the creation of these business services.

## Governing the Service Factory

> ***"To regret one's own experiences is to arrest one's own development. To deny one's own experiences is to put a lie into the lips of one's life. It is no less than a denial of the soul."***
>
> **—Oscar Wilde**

Service transformation planning was one of the capability priorities that SOA governance needed so that it could work with the EA group and help govern the resultant business service priorities. Service development lifecycle controls (SDLCs) represent another area of great concern. After all, the adherence to best practices and a consistency of approach across the development organizations are needed, especially to help enforce the results of the service transformation planning.

### Service Development Lifecycle Controls

Ideation has known for some time that it needs a common and structured development process. Some groups perform peer reviews from time to time on their code, but the reviews tend to be haphazardly completed.

Another group at Ideation is considering a standardized development approach. Your SOA governance team has been asked to propose governance control gates for the service development lifecycle of

- Business requirements
- Solution architecture
- Service identification and specification
- Service build
- Service test
- Service certification

## Control Gates for Ideation

In consultation with the development stakeholders, you decide to create the following control gates for the Ideation SDLC:

- Business Requirements control gate
- Solution Architecture control gate
- Service Design control gate
- Service Build control gate
- Service Test control gate
- Service Certification and Deployment control gate

As a first step, you create the template for the Business Requirements control gate and elicit feedback, as shown in Table 8.2.

**Table 8.2** Ideation Business Requirements Control Gate

| Section | Contains | Definition |
|---|---|---|
| Name | Control gate name | Business Requirements. |
| Motivation | The justification for this control gate. | We must have a consistent and well-formed set of business requirements to create services that give us the greatest amount of agility and reuse. This requires that we have a consistent and repeatable approach to the specification of requirements and that this specification gives the required information for the rest of the development lifecycle. |
| Objective | The output objective | Ensure that the business requirements have a sufficient level of content regardless of form so that development can deliver on the requirement; IT will have a true specification of the business need and objective. This better enables IT to size and estimate this project. Enables the alignment of cross-functional teams on the delivery of the business requirement. |
| Trigger | Triggering event | Complete business requirements are delivered by the business to the project manager. |

**Table 8.2** Ideation Business Requirements Control Gate (continued)

| Section | Contains | Definition |
|---|---|---|
| Scale / Applicability Guidance | Indication of what scale/ style of review is appropriate for the circumstance | Perform if project is > $100K in size. |
| Review type | General classification of review types:<br>■ Peer review<br>■ Submission to approver<br>■ Workshop<br>■ Formal meeting | A workshop to review the requirements artifact and identify that the correct level of detail per the executive design authority standard for business requirements specification has been followed. The following roles are represented at the workshop:<br>Application architect<br>Process analyst<br>Business architecture<br>Business analyst<br>Requirements analyst<br>Solution designer<br>Project manager |
| Concept/approach | Description of how the review is conducted and how it supports the objective | The project manager schedules and leads the workshop.<br>The application architect, business analyst, requirements analyst, and solution designer must validate the requirements while meeting the needs of the next steps of the development lifecycle. That is, they have the information they need to do a good job and the requirements are understood. |
| | | The business architecture must validate the requirements by following the architecture review board (ARB) requirements standard. |
| | | Needed updates to requirements are noted and assigned for follow up, or requirements are approved or disapproved (with updates required). |
| | Governing authority | Business architecture is the governing authority responsible for validating form and that all standards for business requirements are followed. |
| | Responsible/manages | The project manager schedules and leads the workshop. |

| Section | Contains | Definition |
|---|---|---|
| | Accountable - approves/ conditional approval/rejects | The head of line of business (LoB) must be accountable and approve the business requirements with a final signoff. |
| | Consults/supports/performs | The application architect understands the requirements and ensures all the requirements information needed for the high-level solution architecture (the architecture solution document) is there. |
| | | The process analyst answers queries about the business process and ensures that the rest of the requirements conform to the applicable processes. |
| | | The business analyst is responsible for delivering the requirements and is available to answer questions about the requirements or to follow up with any updates needed. |
| | | The requirements analyst understands the requirements, creates the specification, and identifies the specifications applicable to each development group (to understand which areas are impacted). |
| | | The solution designer understands the requirements and will take the architecture solution document from the application architect and create the solution design (end-to-end solution). The solution designer also oversees the components of the solution design. Test/QA—Is this requirement sufficiently articulated to be validated? |
| | Informed | All participants, governance specialist. |
| Artifacts to be made available | The input that is required | Business requirements should be circulated before the workshop to all participants. (A customer service level agreement [SLA] needs to be considered and agreed to.) |
| | | Opinions must be sought from the application architect, business analyst, requirements analyst, and solution designer to validate the requirements. |

**Table 8.2** Ideation Business Requirements Control Gate (continued)

| Section | Contains | Definition |
|---|---|---|
| | What opinions to be sought | The application architect, business analyst, requirements analyst, and solution designer must validate that the requirements meet the needs of the next steps of the development lifecycle. That is, they have the information they need to do a good job, and the requirements are understood. |
| Key review criteria | Highlights the key criteria | The business requirements standard must be adhered to, including the following:<br>■ Consistency of requirements within the business requirements package |
| | | ■ Adherence of business requirements to business processes |
| | | ■ Adherence of business requirements to regulatory and organization compliances and standards |
| Key work product acceptance criteria | Key acceptance criteria to be met for successful review of work product | Consistency and completeness of the business requirements must be followed. Any inconsistencies or gaps or opportunities for improvement or rework of the business requirements must be specified. Record next steps, work assigned, and schedule agreed. Any follow up agreed. |
| Checklist | Sets a suggested process for the review. Provides objective support for recommendations, rework requests, and signoffs. | Business requirements checklist will be used. |
| Escalation process | Name of escalation process to be used to request an exception to the control gate result | Escalates a decision to the program management office (PMO) to drive the business requirements process to conclusion. |
| Measurements | Metrics to be captured during or at the end of this control gate | Governance metrics on this gate, including incrementing the number of times this gate has been implemented, incrementing the number of times this business unit has been through this gate, and the results (pass, fail, pass with conditions), and the checklist score. |

| Section | Contains | Definition |
|---|---|---|
| Outcomes | Who is responsible for final signoff, who gets notified, and who are the results delivered to? | Final signoff—Head of the LoB.<br>Notification—All on the informed list and service registrar.<br>Delivery—The PMO will be the official owner of the results. |

As a next step, you need to create and get feedback on the control gate for the Solution Architecture control gate, as shown in Table 8.3.

**Table 8.3** Ideation Solution Architecture Control Gate

| Section | Contains | Definition |
|---|---|---|
| Name | Control gate name | Solution Architecture |
| Motivation | The justification for this control gate | Approval to proceed with proposed solutions presented in conformance to technology standards and principles:<br>■ To confirm deliverables are encompassing and "fit for purpose"<br>■ To promote deliverable integration and support<br>■ To proactively understand impacts before further investment |
| Objective | The output objective | To assess deliverable's impact on member domains and resulting consideration:<br>■ To agree on integration, interfaces, and support of deliverables<br>■ To ensure alternative approaches and solutions<br>■ To approve new IT solutions presented |
| Trigger | Triggering event | Project manager receives completed solution architecture document. |
| Scale / applicability guidance | Indication of what scale/ style of review is appropriate for the circumstance | Perform if project size > $100K. |

**Table 8.3** Ideation Solution Architecture Control Gate (continued)

| Section | Contains | Definition |
|---|---|---|
| Review type | General classification of review types:<br>■ Peer review<br>■ Submission to approver<br>■ Workshop<br>■ Formal meeting | A workshop to review the requirements artifact and identify that the correct level of detail per the guide lines has been specified. The following roles are needed:<br>Application architect<br>Process analyst<br>Business architecture<br>Business analyst<br>Requirements analyst<br>Solution designer<br>Head of line of business |
| Concept/approach | Description of how the review is conducted and how it supports the objective | The business analyst must validate the requirements as meeting the standards for requirements, and must then trigger the business requirement review workshop to be scheduled and completed. |
| Participants, roles, and interests | Governing authority | ARB. |
| | Responsible/manages | Project manager—Trigger the meeting and provide follow up and results. |
| | Accountable - approves/ conditional approval/rejects | Lead architect—Responsible for presenting and answering questions on the architecture solution document. |
| | Consults/supports/performs | Solution designer—Responsible for representing the technical expertise of standards and policies and evaluating the architecture solution document end-to-end solution as being feasible.<br>Business analyst—Responsible for representing the business client and validates that the solution meets needs of the business and accepts or declines proposed trade-offs.<br>Architecture executives—Provide guidance, review output, and ensure governance.<br>Infrastructure architect (if heavy infrastructure in solution)—Have provided infrastructure architecture in the architecture solution document and answer questions as needed. |

| Section | Contains | Definition |
|---|---|---|
| | | Infrastructure designer—Validate the solution architecture for structural feasibility. |
| | | Requirements analyst—Review and understand the high-level solution architecture and context of downstream requirements and analyst comments. |
| | | Data architect—Provided data architecture in the architecture solution document and answers questions as needed. |
| | Informed | All participants, Center of Excellence (CoE). |
| Artifacts to be made available | The input that is required | Architecture solution document |
| | What opinions to be sought | The solution designer, solution architect, infrastructure designer, and the requirements analyst must validate the architecture solution document as meeting the needs of the next steps of the development lifecycle. That is, they have the information they need to do a good job, and the high-level design is understood. |
| Key review | Highlights the key criteria | The architecture solution document criteria must comply with architectural standards, policies, and patterns. |
| | | The architecture solution document conforms to existing application future views where those views exist. |
| | | The architecture solution document highlights key infrastructure impacts and requirements. |
| | | Have services been identified and classified, and are they at the correct level of granularity (services litmus tests)? |
| Key work product acceptance criteria | Key acceptance criteria to be met for successful review of work product | Consistency and completeness of the high-level design must be followed. Any inconsistencies or gaps or opportunities for improvement or rework of the high-level design must be specified. Record next steps, work assigned, and schedule agreed. Any follow up agreed. |

**Table 8.3** Ideation Solution Architecture Control Gate (continued)

| Section | Contains | Definition |
|---|---|---|
| Checklist | Sets a suggested process for the review. Provides objective support for recommendations, rework requests, and signoffs. | Follow the architecture solution document checklist. |
| Escalation process | Name of escalation process to be used to request an exception to the control gate result | Exceptions to demands for rework can be approved by the ARB. Where the issue is a dispute on service identification, the ARB shall send that issue to the CoE for resolution. |
| Measurements | Metrics to be captured during or at the end of this control gate | Governance metrics on this gate, including incrementing the number of times this gate has been implemented, incrementing the number of times this lead architect has been through this gate, the results (pass, fail, pass with conditions), and the checklist score |
| Outcomes | Who is responsible for final signoff, who gets notified, and who are the results delivered to? | Final signoff—ARB lead. Notification—All participants are notified, service registrar. Handoff—The PMO will be the official owner of the results. |

# Implementing the SOA Governance Model

> ***"Organization can never be a substitute for initiative and for judgment."***
>
> **—Louis Brandeis**

After due consideration, Ideation leadership believes that it would have a better forum for initiative and judgment by implementing two new organizations. These include an SOA Center of Excellence (CoE) and an SOA executive review board (ERB). An architecture review board (ARB) already exists, but company leadership recognizes that the three groups will need to work with each other and that governance must define how.

## Service Processes, Organizations, Roles, and Responsibilities

The first thing addressed is a mission statement for each of the two new organizations.

The CoE has the following mission statement:

The SOA Center of Excellence (CoE) is an IT cross-organizational team that has been established at Ideation to support and enable SOA governance and ensure the institutionalization of SOA. The CoE will specifically be responsible for the following:

- The CoE will be the implementation mechanism of SOA governance and is responsible for identifying and enabling the processes, standards, policies, mechanisms, roles, and responsibilities for SOA governance.
- The CoE will work with the ARB to make sure that the necessary SOA patterns, policies, and standards are created, implemented, and maintained.
- The CoE will provide governance mechanisms and metrics for the quality review of SOA development, certification, and operations, and will validate that good governance is taking place with vitality.
- The CoE is the center of adoption, change, and vitality of the SOA architecture. The CoE will enable the communication and socialization of the CoE, SOA architecture, and architecture concepts as approved by the ARB.
- The CoE will identify the SOA skills gaps and training programs needed and will provide follow up to validate that training is successful and vital.
- The CoE will be a center of SOA skills and will consult on SOA projects and provide guidance on usage of SOA technologies.
- The CoE will provide an SOA registry/repository, and the CoE will manage the governance, control, and deployment of all operational services in the registry/repository.
- The CoE will ensure the project, architecture, and service assets are harvested and reused where appropriate.

The ERB has the following mission statement:

The SOA executive review board (ERB) is a linking mechanism that integrates business and IT with the appropriate stakeholders. Its purpose is to make key strategic and tactical decisions concerning the SOA journey and to investigate and resolve issues that require cross-organizational and business/IT decisions. It is a business and IT cross-organizational and cross line-of-business (LoB) team.

The ERB sets the direction and pace for services adoption and resolves or recommends solutions for exception requests, including business domain issues such as ownership, funding, and reuse policy. The board performs reviews of proposed investments to ensure commonality opportunities and reuse investments are considered.

- The ERB will resolve any issue needing resolution for an SOA project that involves a cross-organizational or cross-LoB issue. The ERB will do this in a timely fashion.
- The ERB will be the controlling organization for the strategic direction of creating business agility.
- The ERB will be the last review for any exception requests from an SOA program.
- The ERB will be the decision maker on all investments for the portion of the IT budget that is allocated for business agility.
- The ERB will be the primary communication mechanism for business agility and will work across both business and IT as necessary.
- The ERB will provide oversight and direction for the SOA Center of Excellence (CoE) and the architecture review board (ARB).

An organizational diagram was created that summarizes this arrangement (see Figure 8.4).

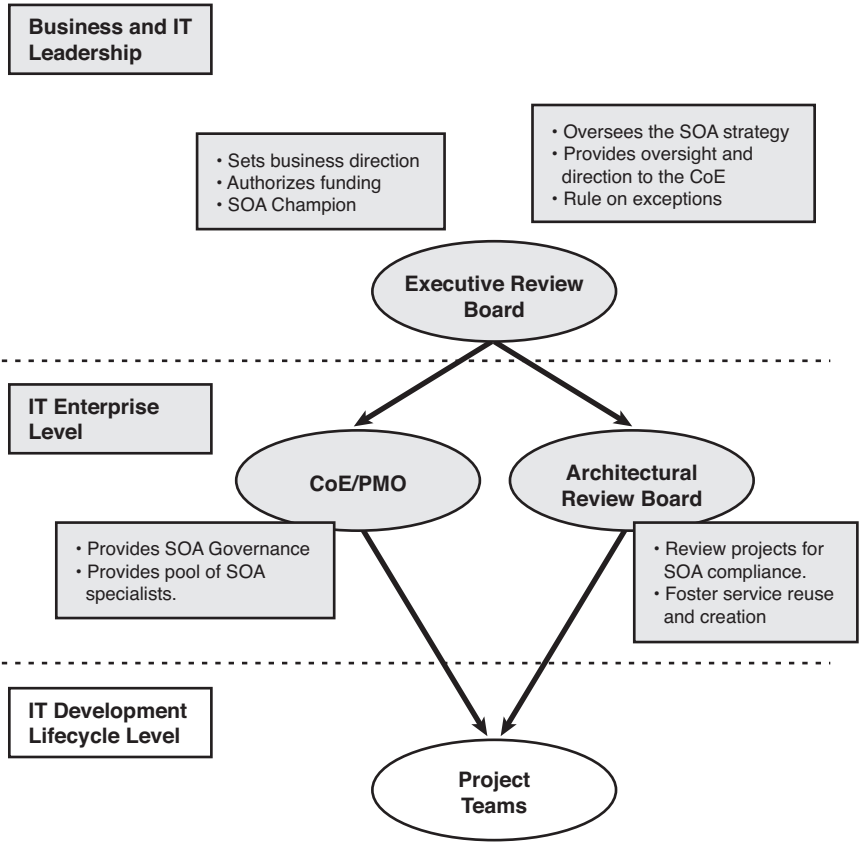



**Figure 8.4** Ideation Organizational Chart for SOA Governance

The first SOA project for Ideation is a pilot, and things go well. On the other hand, there is no need for governance for the pilot project itself, because all the work is performed within a single organization. Several programs, which are enterprisewide and cross LOB and IT department fiefdoms, are up next.

## Service Ownership and Funding

Almost immediately, an issue manifests itself that the ERB must resolve: SOA ownership and funding. All communication companies have a function called a rating, which provides for a specific price to be

attached to a specific telecommunications transaction or set of transactions. As mentioned in the "Building the Service Factory" section of the Case Study, Ideation has a tactical business goal of creating a single rating function that spans all the LOB. As Ideation has moved to offer more product bundling, the different pricing structures have led to a chaotic situation. In some cases, different pricing was available on the same service from a different sales group within the company. Consequently, one of the second-phase business agility projects is to create an SOA business service with the name Create Rating.

During the requirements phase, the service business analysts found that the different groups from the various LOB would not cooperate. Information was withheld, meetings were missed, and it soon became clear that no amount of cajoling was going to get the team to a satisfactory solution.

The item was raised by the PMO via the CoE to the ERB. The business vice-president of operations, Gus Lee, is assigned to lead a group to address and resolve the requirements for this function. Gus has no direct involvement in rating, and was therefore considered a neutral business executive. In addition, Gus has an IT background from his time in the Army, and could work productively across business and IT. Gus selected one IT and one business colleague from the ERB, but also filled out his "tiger team" with an SME from an IT group that he believed would give him unbiased information.

Gus is busy running operations on a day-to-day basis, but he diligently holds meetings for one hour each day and made and followed up on assignments with his team. It soon became clear that this was not a technical or business issue, but a political one. The business director of each of the rating teams felt threatened by this business agility initiative, and based on past experience, thought that they could just wait out this passing storm and return to business as usual. Gus personally interviewed each of these leaders. His ERB colleagues on the tiger team were able to summarize statistics that showed that Ideation was losing about $50 million a year in revenue due to rating mistakes and another $100 million a year in opportunity cost due to an inability to respond to bundle-product pricing requests in a timely manner. The SME produced critical information about how each of the rating groups actually operated and how IT supported them with procedural code that was highly customized.

Gus assembled this information and then scheduled a meeting with the Ideation chief financial officer (CFO). The CFO pointed out that the $100 million "opportunity cost" was a soft number that he did not agree with, but the $50 million in lost revenue was well documented and could not be refuted. A week later, he called Gus into his office and told him that he was reorganizing all the rating groups into one organization immediately, and that David Croslin had been assigned to clean up the mess. Further, David had committed to providing timely requirements on the Create Rating service for the PMO and that he personally would review them and sign off.

The results were reported to the ERB at its next meeting, and the tiger team was duly terminated with a "job well done." The total time for the issue to be resolved was one month. The ERB had an SLA of three weeks for this type of issue, but it was agreed ahead of time that this was a more complicated issue and the SLA would be relaxed by several weeks. Updates had been discussed at the weekly ERB meeting, and it was agreed that this pattern of issue resolution had worked well and should be repeated in the future.

## Managing the Service Lifecycle

> ***"A pessimist sees the difficulty in every opportunity; an optimist sees the opportunity in every difficulty."***
>
> **—Winston Churchill**

Although the SOA pilot had been judged a success, one area of difficulty is the service certification. Truth be told, from an IT governance point of view, Ideation has never had a smooth handoff procedure from development and test to operations. The operations group complains continuously that programs are dumped on their department without the necessary documentation and without their approval.

### Service Certification

Gus Lee, vice-president of operations, gained significant credibility based on his success in solving the rating problem. In conjunction with the CoE leader, Ike Elliott, they decided to bring before the ERB the

problem of service certification. Both had reviewed the process used during the SOA pilot and pronounced it "severely inadequate." IT had been able to ignore operations in the past, but because of the governance that was being put in place and the increasing power of the ERB, they were no longer able to do so. Tom Baker, an ace service architect on the CoE, was asked by Gus and Ike to lead a tiger team to solve the service certification problem. His instructions were to find a methodology for adding a new "service" entity to the set of enterprise assets that operations would support and which would be readily available and documented to potential service consumers.

Based on his experience, Tom recognized that although this was not an entirely new step within IT operations procedures, there were interoperability challenges new to services that had to be addressed. He believed that technology standards and tools (registry, repository, and so on) were quickly evolving and might be useful here. Tom drafted two SMEs in the area of requirements and operations to assist him. To understand the additional accountability requirements that were needed, the SMEs interviewed stakeholders in operations, business analysis, and potential service consumers for the rating service that is being built. They reported the results to Tom and further recommended that service certification should be carried out under the control of the Ideation Quality Assurance department. This process needed to include various details that are developed and documented during the service design and development phase, such as SLA and QoS. It is believed by Tom and his team that this final certification review step would formalize the QA process before physically publishing the service as being "enterprise ready," with an assured QoS and a full and complete set of support materials. This would also provide approval for deployment to production and update of the service registry and repository.

Their published requirements identified that certification ensured information such as a version number, ownership, who is accountable for the service, and classification and availability of the service be published as mandatory service metadata. They also required that documentation be provided that helps the IT operations staff understand the service and be able to support it for all levels of operation (first through fourth). Requirements for certification need to ensure the correctness of service contracts created as part of service specification and that it could be demonstrated that the service has adhered to the required sequence of steps for the service lifecycle status, including intermediate peer reviews.

The purpose of defining formal service statuses and allowed status transitions is to clearly provide information related to the stage in which the service is at a given point in time. The certification review results in a pass or fail status for the service in consideration. Pass status indicates that the service is ready to be published. Fail status indicates that further work needs to be done before this can be certified as a service.

It is recommended that the SOA CoE assist IT operations with the service certification during the adoption phase of SOA.

As a result of these requirements, Tom and his team create the service certification checklist shown in the following tables. The final report is reviewed and accepted by the ERB and then implemented via the PMO and the CoE.

The Ideation Service Certification Header is shown in Table 8.4.

**Table 8.4** Ideation Service Certification Header

| Certification Item | Certification Explanation |
|---|---|
| Description | Ensure that each service passes all necessary certification criteria before it is deployed to production. |
| Governance goal | Reduce risk. |
| Governed entities | Type: Deployed services. |
| Inputs | Deployed services, test results. |
| When to apply | This checklist should be completed before any service is deployed to production. |

The Ideation service certification detail is listed here.

### Ideation Service Certification Detail

- ☐ Functional test plan is at a level of rigor and detail appropriate for a core software asset?
- ☐ Functional test plan has been completed satisfactorily?
- ☐ Nonfunctional testing, including performance testing, iscomplete?
- ☐ Service meets security architecture requirements?
- ☐ Risk assessment conducted according to existing risk management framework. Where the risk assessment shows a residual risk greater than LOW, approval for deployment of the service must be sought from security architect. Residual risk of EXTREME must be referred to CIO for approval.

- ☐ Performance engineering—design of the service has involved those parties with responsibility for production support of all platforms that the service touches and all agree that the service can be deployed without adversely affecting performance.
- ☐ Performance engineering team has been engaged and agree the service levels for availability can be met.
- ☐ Response times adequate and fit within a broader end-to-end performance budget.
- ☐ Deployment plan is complete (need to review the existing template and augment where necessary)?
- ☐ Does the operational area know how to deploy the service?
- ☐ Are all the dependencies identified in the plan?
- ☐ Organizational issues related to maintenance of the service have been resolved?
- ☐ Does the operational area know where errors (fatal, warning, informational) will be reported at runtime?
- ☐ Does the operational area know how to react to runtime errors?
- ☐ Is the service (or supporting runtime environment) able to report that it's under stress?

## Governance Vitality

> ***"Nature knows no pause in progress and development, and attached her curse on all inaction."***
>
> **–Johann Wolfgang von Goethe**

Ike Elliott, the Ideation CoE Lead, is charged with creating metrics that will assist in measuring the progress of creating business agility and providing vitality measurements that can enable the CoE to change the governance processes when needed.

## Service Governance Vitality

First, Ike reviews the business goals from the original study chartered by the Ideation board of directors, restated here:

- Be able to create new product bundles in 6 months' time measured from product marketing ideation to product offering in the marketplace.
- Decrease the operating run rate by 10%.
- Create upsell opportunities with our current customers that result in an increased average revenue per user (ARPU) by 5%.
- Create a common services organization and merge duplicative corporate capabilities into this single organization. This will include finance, HR, and data center support.
- Create and implement a business agility capability that includes a governance organization and appropriate business-IT linking groups to work with enterprise architecture, the business, and IT and bring quality to Ideation's business and IT.

The last item is satisfied by the implementation of the CoE and the ERB, and it is agreed that these were performing well. In particular, the metric of measuring how long it took to resolve issues brought before the ERB had been an excellent way to keep the focus on resolving issues expeditiously (with an average of three weeks). The planning for the common services organization has been ongoing for several months, and an announcement from the CEO was scheduled to be made next week.

The first three items from the board are quite strategic and need some tactical planning around them to be meaningful. To this end, Ike assembled a team that included Lance Miller, chief architect and head of the architecture review board (ARB), two key subject matter experts from the ARB, and Katherine Bull, vice-president of finance and a standing member of the ERB.

For example, the requirement for the run-rate cost reduction required Katherine to work with Lance to identify the tactical means of such a reduction. As a result of this analysis, the following metrics are created by Katherine to measure the plan for decreasing the operating run rate by 10%:

- METRIC: Percentage of automated flow through for provisioned orders
  - DEFINITION: Percentage of provisioning orders provisioned by fulfillment function without human intervention ("no-touch orders")
  - GOAL: 75% automated flow through for provisioning by end of year (estimated $111M savings)
    - SUBGOAL: 90% of disconnect orders automated
    - SUBGOAL: 80% of supplement orders automated
    - SUBGOAL: 95% of orders with all correct information at initial order release

- METRIC: Percentage of no-touch orders for operations
  - DEFINITION: Percentage of network operations requests from trouble tickets that can be responded to without human intervention
  - GOAL: 50% no-touch orders for operations by end of year (estimated $17M savings)

- METRIC: Percentage of repairs that can be performed remotely
  - DEFINITION: Percentage of repair requests that can be performed without a truck roll by using remote and automated capabilities to resolve network operations requests from trouble tickets that can be responded to without human intervention
  - GOAL: 30% remote repair for operations by end of year (estimated $39M savings)

While Katherine and Lance continue this pattern to create metrics for the other two Ideation strategic goals mentioned previously, Ike is working with some of his CoE governance specialists to analyze the areas within the implemented governance processes where vitality is a concern. In particular, Ike wants to make sure that the right metrics are in place for the service development lifecycle so that the CoE can adjust the tightness of governance for the various control points.

Ike creates multiple metrics for each of the control points, with two of those noted here:

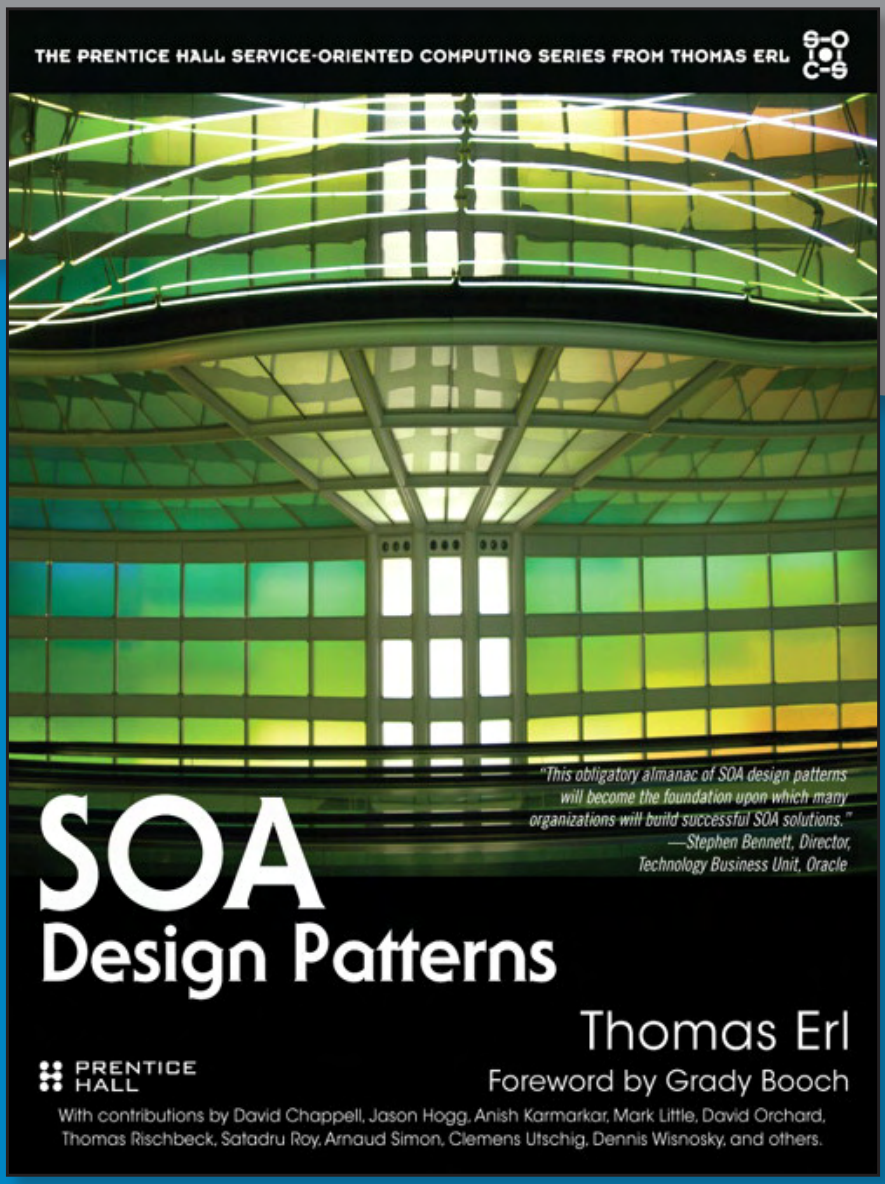
- METRIC: Percentage of development trouble tickets caused by inadequate requirements
  - DEFINITION: Development trouble tickets are assigned a root cause with "inadequate requirements" being one of the root causes. To test the efficacy of the business requirements control point, measure the percentage of all tickets caused by bad requirements.
  - GOAL: After establishment of a six-month baseline, expect this percentage to continually decrease.
  - VITALITY: Further analyze the causes of inadequate requirements and make and implement changes to the business requirements control point as needed.

- METRIC: Number of violations of architecture standards, policies, and reference architectures in solution architecture
  - DEFINITION: The solution architecture control point checks compliance with the architecture standards, policies, and reference architectures as created and approved by the ARB. In the review, the number of violations will be tracked.
  - GOAL: After establishment of a six-month baseline, expect this percentage to continually decrease.
  - VITALITY: Further analyze the causes of violation and decide whether the policies, standards, and reference architectures are inadequate, or whether further training needs to take place to make the solution architects more conversant with them.

# Conclusion

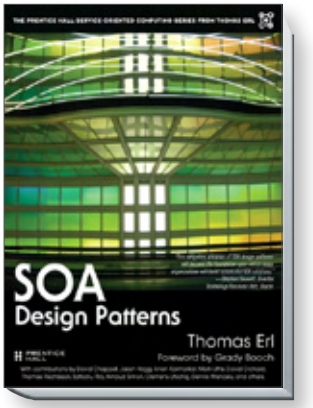
It's been six months now since you took over as head of the SOA governance function. During that time, you've initiated an SOA governance planning assessment and addressed the governance of the areas of service transaction planning, the service development lifecycle controls, service processes, organization, roles and responsibilities, service ownership and funding, service certification, and service governance vitality.

Six months ago, you never would have believed that you could have accomplished so much. Good thing you bought that great book on SOA governance and applied everything you learned!



Thomas Erl

BUY ME




# SOA Design Patterns

**"*SOA Design Patterns* is an important contribution to the literature and practice of building and delivering quality software-intensive systems."**

— GRADY BOOCH, IBM Fellow

In cooperation with experts and practitioners throughout the SOA community, best-selling author Thomas Erl brings together the de facto catalog of design patterns for SOA and service-orientation. More than three years in development and subjected to numerous industry reviews, the 85 patterns in this full-color book provide the most successful and proven design techniques to overcoming the most common and critical problems to achieving modern-day SOA.

Through numerous examples, individually documented pattern profiles, and over 400 color illustrations, this book provides in-depth coverage of:

- Patterns for the design, implementation, and governance of service inventories — collections of services representing individual service portfolios that can be independently modeled, designed, and evolved.
- Patterns specific to service-level architecture which pertain to a wide range of design areas, including contract design, security, legacy encapsulation, reliability, scalability, and a variety of implementation and governance issues.
- Service composition patterns that address the many aspects associated with combining services into aggregate distributed solutions, including topics such as runtime messaging and message design, inter-service security controls, and transformation.
- Compound patterns (such as Enterprise Service Bus and Orchestration) and recommended pattern application sequences that establish foundational processes.

The book begins by establishing SOA types that are referenced throughout the patterns and then form the basis of a final chapter that discusses the architectural impact of service-oriented computing in general. These chapters bookend the pattern catalog to provide a clear link between SOA design patterns, the strategic goals of service-oriented computing, different SOA types, and the service-orientation design paradigm.

AVAILABLE

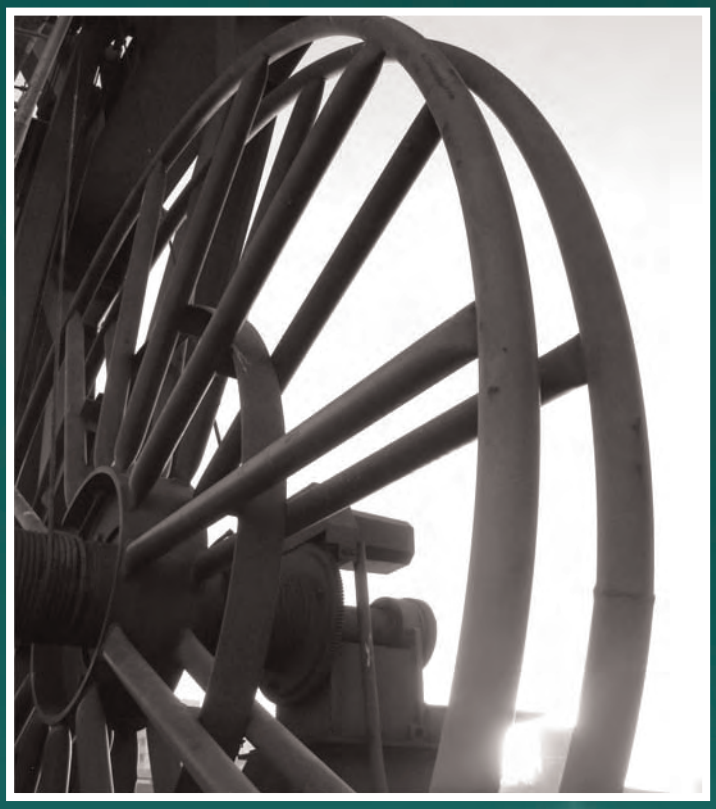
- BOOK: 9780136135166
- SAFARI ONLINE Safari Books Online
- EBOOK: 0132361191
- KINDLE: B00139VU0Q

## About the Author

THOMAS ERL is the world's top-selling SOA author, *Series Editor of the Prentice Hall Service-Oriented Computing Series* from Thomas Erl, and Editor of The *SOA Magazine* (www.soamag.com).With over 100,000 copies in print world-wide, his books have become international bestsellers and have been formally endorsed by senior members of major software organizations, such as IBM, Microsoft, Oracle, BEA, Sun, Intel, SAP, CISCO, and HP. Thomas is the founder of SOA Systems Inc. and the internationally recognized SOA Certified Professional program (www.soacp.com). Articles and interviews by Thomas have been published in numerous publications, including *The Wall Street Journal* and *CIO Magazine*.

PRENTICE HALL

**informit.com/ph**



# Chapter 12

## Service Implementation Patterns

Service Façade

Redundant Implementation

Service Data Replication

Partial State Deferral

Partial Validation

UI Mediator

Each of the following design patterns impacts or augments the service architecture in a specific manner, thereby affecting its physical implementation. Most are considered specialized, meaning that they are to be used for specific requirements and may not be needed at all (and are rarely all used together).

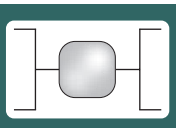
Arguably the most important pattern in this chapter is Service Façade (333) because it introduces a key component into the service architecture that can help a service evolve in response to on-going change.

Redundant Implementation (345), Service Data Replication (350), and Partial State Deferral (356), on the other hand, help establish a more robust service implementation by addressing common performance and scalability demands. Unlike Service Façade (333), which actually alters the complexion of the service architecture, these three patterns can be more easily applied subsequent to a service's initial deployment.

Partial Validation (362) is a consumer-focused pattern that helps optimize runtime message processing and UI Mediator (366) is a pattern specialized to bridging potential usability gaps between services and the presentation layer.

## Service Façade

*How can a service accommodate changes to its contract or implementation while allowing the core service logic to evolve independently?*



| | |
|---|---|
| **Problem** | The coupling of the core service logic to contracts and implementation resources can inhibit its evolution and negatively impact service consumers. |
| **Solution** | A service façade component is used to abstract a part of the service architecture with negative coupling potential. |
| **Application** | A separate façade component is incorporated into the service design. |
| **Impacts** | The addition of the façade component introduces design effort and performance overhead. |
| **Principles** | Standardized Service Contract, Service Loose Coupling |
| **Architecture** | Service |

**Table 12.1**
Profile summary for the Service Façade pattern.

### Problem

A given service will contain a core body of logic responsible for carrying out its (usually business-centric) capabilities. When a service is subject to change either due to changes in the contract or in its underlying implementation, this core service logic can find itself extended and augmented to accommodate that change. As a result, the initial bundling of core service logic with contract-specific or implementation-specific processing logic can eventually result in design-time and runtime challenges.

For example:

- A single body of service logic is required to support multiple contracts, thereby introducing new decision logic and requiring the core business routines to process different types of input and output messages.

- The usage patterns of shared resources accessed by the service are changed, resulting in changes to the established service behavior that end up negatively affecting existing service consumers.
- The service implementation is upgraded or refactored, resulting in changes to the core business logic in order to accommodate the new and/or improved implementation.
- The service is subjected to decomposition, as per Service Decomposition (489).

Figure 12.1 illustrates the first of these scenarios.

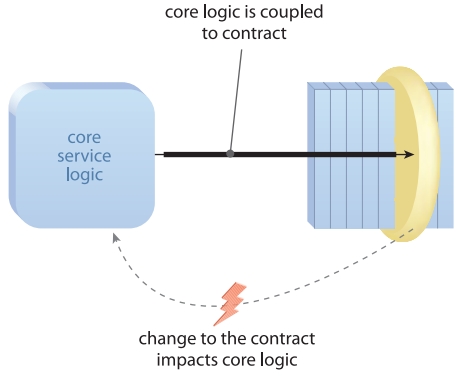



**Figure 12.1**
If the core service logic is coupled directly to the contract, any changes to how the service interacts with consumers will require changes to the core service logic. (This represents only one of several of the problem scenarios addressed by this pattern.)

## Solution

Façade logic is inserted into the service architecture to establish one or more layers of abstraction that can accommodate future changes to the service contract, the service logic, and the underlying service implementation (Figure 12.2).

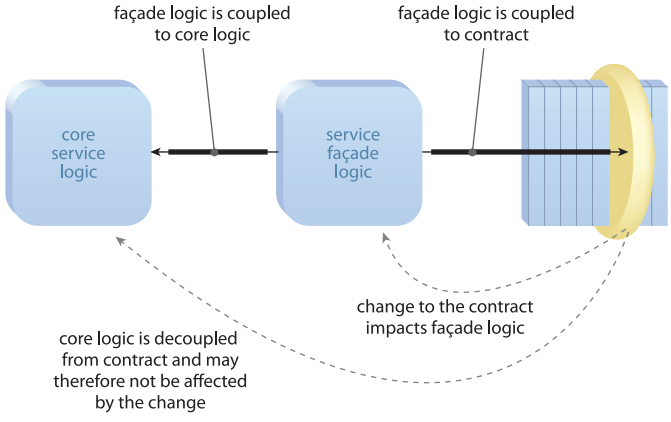



**Figure 12.2**

Façade logic is placed in between the contract and the core service logic. This allows the core service logic to remain decoupled from the contract.

**NOTE**

Service Façade is a versatile pattern that can be applied in many different ways within a given service architecture. The upcoming *Application* section explores several possible application scenarios to demonstrate how service façade logic can be potentially utilized. This section is therefore noticeably longer than the average *Application* section for other patterns.

## Application

Service façade logic is considered part of the overall service logic but distinct from the core service logic, as follows:

- The core service logic is expected to provide the range of functions responsible for carrying out the capabilities expressed by the service contract.
- The service façade logic is primarily responsible for providing supplemental, intermediate processing logic in support of the core service logic.

Service façade logic is generally isolated into a separate component that is part of the service architecture. Common types of logic that tend to reside within a service façade component include:

- *Relaying Logic* – The façade logic simply relays input and output messages between the contract and the core service logic or between the core service logic and other parts of the service architecture. For examples of this, see the descriptions for Proxy Capability (497) and Distributed Capability (510).
- *Broker Logic* – The façade logic carries out transformation logic as per the patterns associated with Service Broker (707). This may be especially required when a single unit of core service logic is used together with multiple service contracts, as per Concurrent Contracts (421).
- *Behavior Correction* – The façade logic is used compensate for changes in the behavior of the core service logic in order to maintain the service behavior to which established consumers have become accustomed.
- *Contract-Specific Requirements* – When service facades are coupled to contracts in order to accommodate different types of service consumers, they can find themselves having to support whatever interaction requirements the contracts express. This can include special security, reliability, and activity management processing requirements. While all of this processing can also be located within the core service logic, it may be desirable to isolate it into façade components when the processing requirements are exclusive to specific contracts.

Service façade components can be positioned within a service architecture in different ways, depending on the nature and extent of abstraction required. For example, a façade component can be located between the core service logic and the contract. Figure 12.3 elaborates on the problem scenario first introduced in Figure 12.1 by showing how a design based on core service logic coupled to the contract can lead to restrictive architectures after multiple contracts enter the picture.

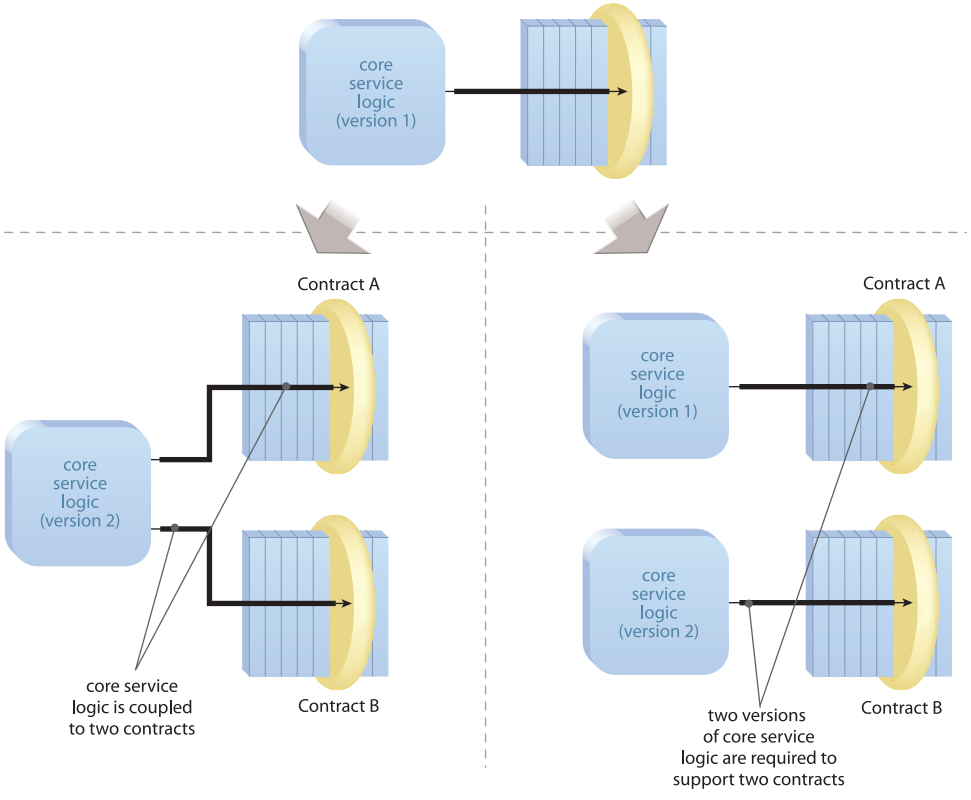



**Figure 12.3**

When designing services we are encouraged to tailor underlying service logic in support of independently customized and standardized service contracts. This results in a high level of logic-to-contract coupling, while allowing the contract itself to be decoupled from its implementation. Although this is considered desirable from a contract coupling perspective, it can lead to undesirable design options for supporting multiple service contracts with just a base unit of core service logic. The first architecture (left) requires a new version of the core service logic that is now coupled to two contracts, while the second (right) requires the creation of a new service altogether, leading to redundant core service logic.

Figure 12.4 illustrates how the abstraction achieved through the use of service façade components allows for the addition of multiple service contracts without major impact to the core service logic. Service façade components are intentionally tightly coupled to their respective contracts, allowing the core service logic to remain loosely coupled or even decoupled. What this figure also highlights is the opportunity to position consumer-specific service logic as an independent (and perhaps even reusable) part of the service architecture.

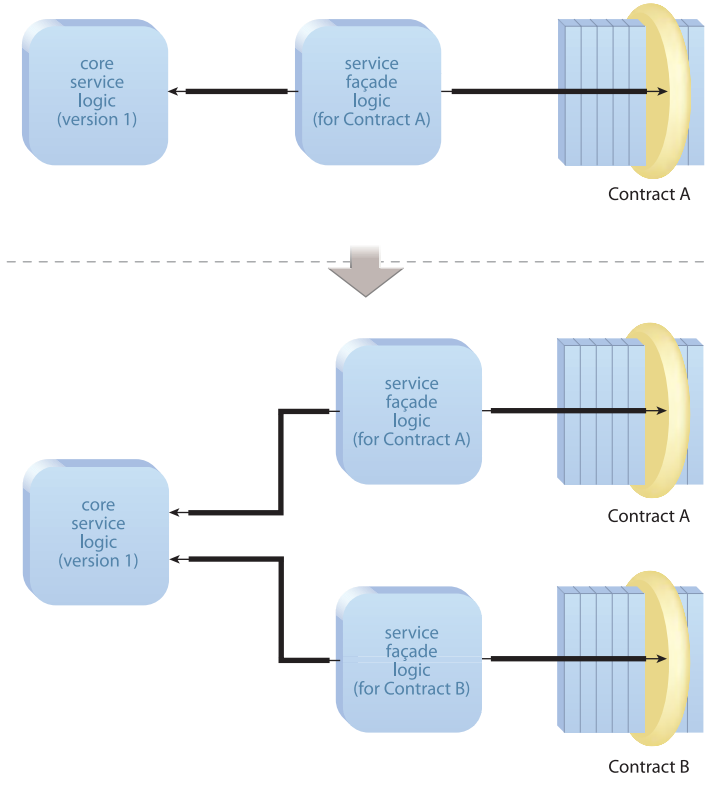



**Figure 12.4**
New service contracts can be accommodated by repeating this pattern to introduce new façade components, thereby potentially shielding the core service logic.

When service façade logic is used to correct the behavior of changed core service logic, it is also typically positioned between the contract and core service logic. This allows it to exist independently from logic that is coupled to (and thereby potentially influenced by) specific parts of the underlying implementation.

Figure 12.5 shows how core service logic coupled to both the implementation and the contract may be forced to pass on changes in behavior to established service consumers. Figure 12.6 then demonstrates how a layer of service façade processing can be designed to regulate service-to-consumer interaction in order to preserve the expected service behavior.

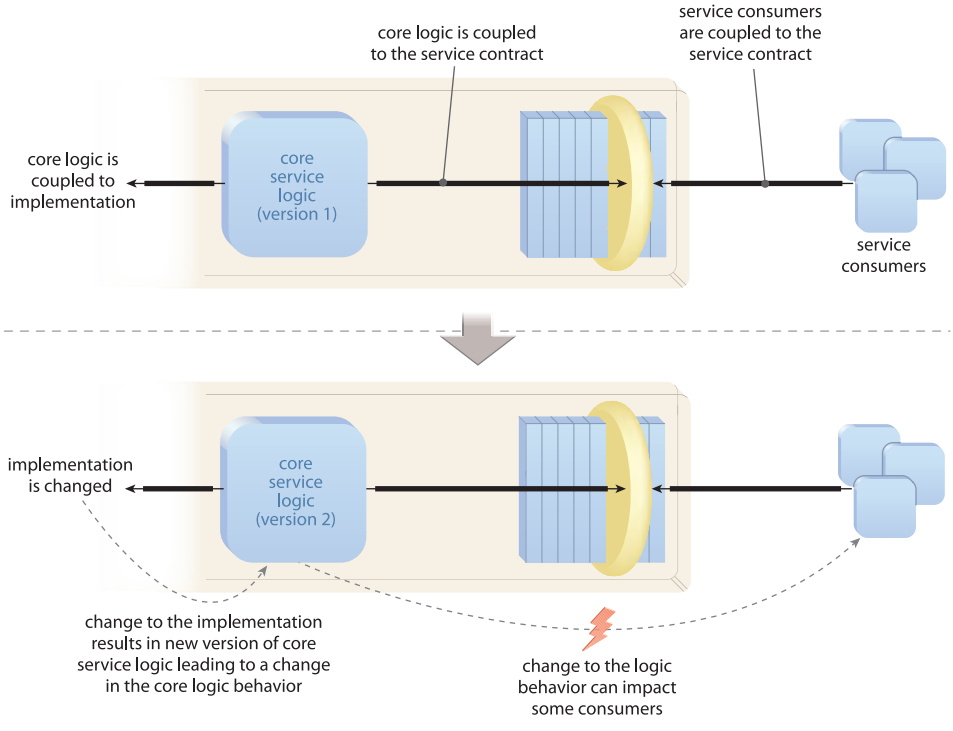



**Figure 12.5**

Parts of the implementation encapsulated and bound to by version 1 (top) of the core service logic are subject to change, resulting in the release of a second version (bottom) that brings with it a noticeable change in behavior. Service consumers coupled to the service contract are affected by this change because the new core service logic is also directly coupled to the contract.

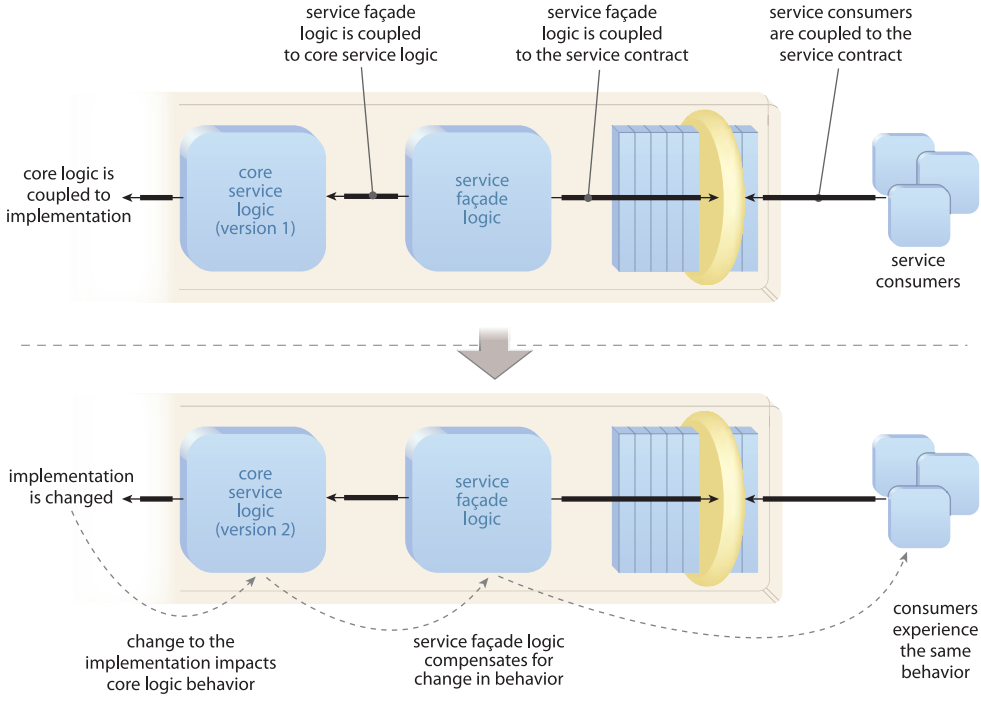



**Figure 12.6**

The core service logic is updated due to the change in implementation, but the behavioral changes are caught by the service façade component which contains additional routines to preserve the original behavior while still interacting with version 2 of the core service logic.

Finally, it is worth pointing out that service façade logic is not limited to acting as an intermediary between the core service logic and the service contract. Figure 12.7 shows an architecture in which components are positioned as facades for underlying implementation resources. In this case, this pattern helps shield core service logic from changes to the underlying implementation by abstracting backend parts.

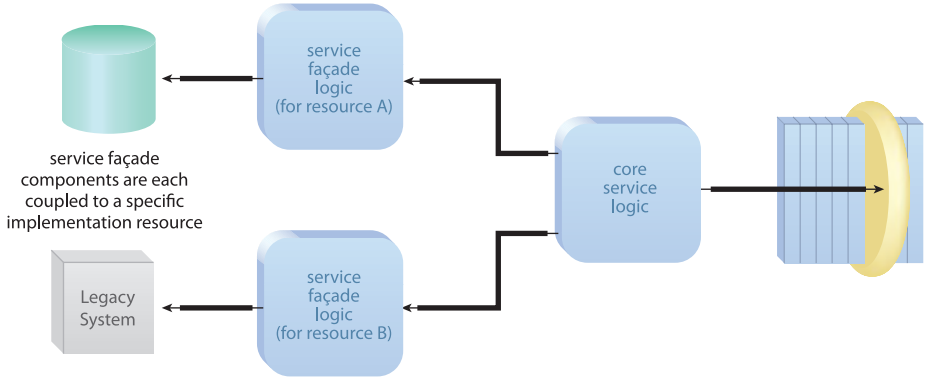



**Figure 12.7**

One service façade component abstracts a shared database (resource A) whereas another abstracts a legacy system (resource B). This abstraction helps protect the core service logic from changes to either of these parts of the underlying service implementation.

## Impacts

Creating façade components results in an increased amount of physical logic decomposition. This naturally introduces additional design and development effort, as well as extra cross-component communication requirements. Although some performance overhead is expected, it is generally minor as long as façade and core service components are located on the same physical server.

Some governance overhead can also be expected, due to the increased amount of components per service.

## Relationships

The structural solution provided by Service Façade helps support the application of several other patterns, including Service Refactoring (484), Service Decomposition(489), Proxy Capability (497), Agnostic Sub-Controller (607), Inventory Endpoint(260), Distributed Capability (510), Concurrent Contracts (421), and Contract Denormalization (414). This pattern is ideally combined with Decoupled Contract (401) in order to provide the maximum amount of design and refactoring flexibility throughout a service's lifespan.

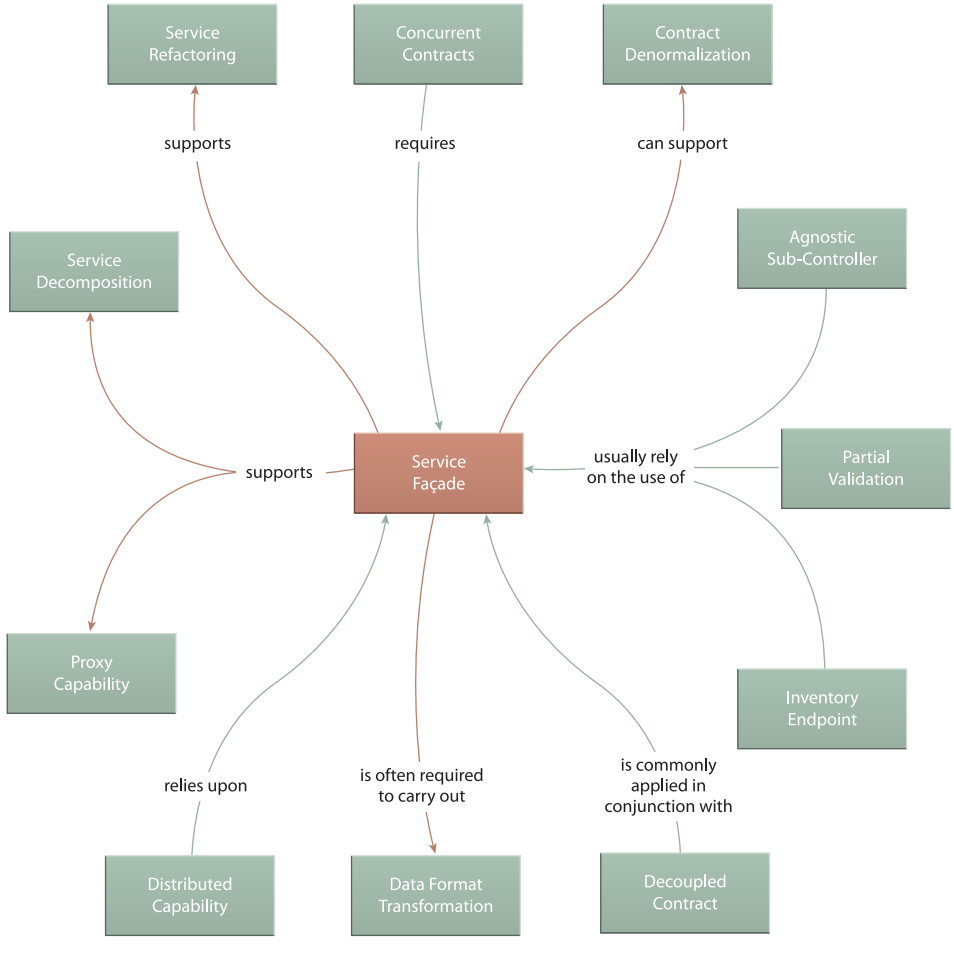



**Figure 12.8**

Service Façade establishes a key part of the service logic that ends up supporting several other service design patterns.

**CASE STUDY EXAMPLE**

The FRC is developing an entity service called Appealed Assessments, which is dedicated to producing a range of reports related to already assessed claims that have been successfully or unsuccessfully appealed. Depending on the nature and scope of the requested report, this service may need to access up to six different repositories in order to gather all of the required data.

A component-based architecture already exists in which a separate wrapper utility component has been created to represent and provide standardized access to all six repositories. This Data Controller component provides all the logic required to fulfill the capabilities of the planned Assessment Reports service in addition to several other generic data access and reporting functions.

Instead of creating new logic to accomplish the same data access tasks, FRC wants to use the Data Controller component as the core service logic for the Appealed Assessments service. However, they are told by the group that owns this component that it can't be altered in support of this service. Furthermore, the component is expected to undergo some changes in the near future that may result in it having to support one additional database plus accommodate the planned consolidation of two existing legacy repositories. As a result, the component needs to remain an independently governed part of the architecture.

The FRC architects decide to design a service façade component that will be used to bind to the official WSDL contract for the Appealed Assessments service. The façade component is appropriately named Data Relayer, and its primary responsibility is to receive service consumer requests via the standardized WSDL contract, relay those requests to the corresponding internal wrapper components, and then relay the responses back to the service consumer.

The Data Relayer component contains a modest amount of logic, most of which is focused on validating data reports received from the Data Controller component and (if necessary) converting them to the format and data model required by the Appealed Assessments service's WSDL `message` construct and associated XML schema complex types.

The resulting service architecture (Figure 12.9) allows the original Data Controller component to evolve independently while establishing the Data Relayer component as an intermediate façade dedicated to the Appealed Assessments service.

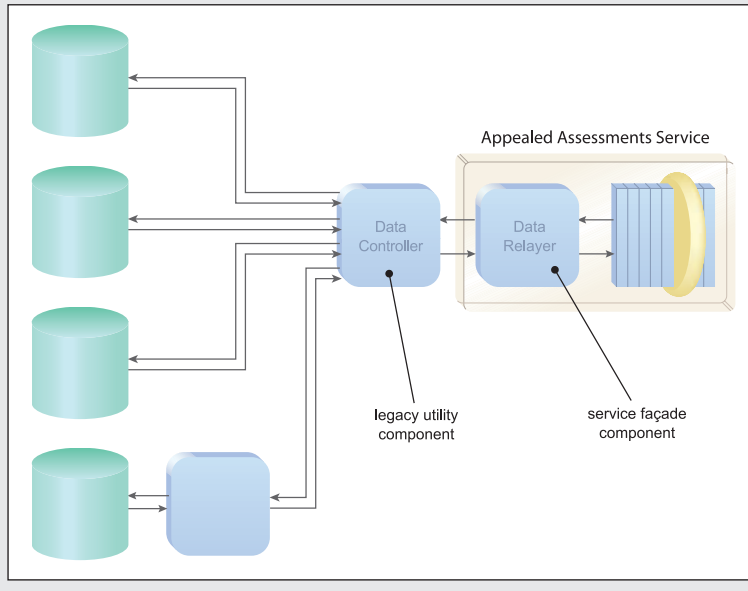



**Figure 12.9**

The Data Relayer service façade component is designed into the architecture of the Appealed Assessments service. Note the bottom database is accessed via a separate API component. This environment (called "MainAST103") is explained in the Legacy Wrapper (441) case study example.

This architecture further accommodates the expected changes to the Data Controller component. Should any of these changes affect the format or data of the reports generated by the Data Controller functions, the Data Relayer component can be augmented to compensate for these changes so that the Appealed Assessments service contract remains unchanged and so that consumers of this service remain unaffected.

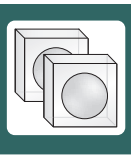
The case study example for Legacy Wrapper (441) continues this scenario by introducing the need to add a legacy wrapper component into the Appealed Assessments service architecture.

**NOTE**

You may have noticed that in Figure 12.9 Data Controller is further labeled as a "legacy" component. This is because even though it is conceptually similar to a utility service, it is an older component that has not been subjected to service orientation. It therefore is not considered a member of the service inventory but is instead (from an SOA perspective) a part of the legacy environment.

## Redundant Implementation

*How can the reliability and availability of a service be increased?*



| | |
|---|---|
| **Problem** | A service that is being actively reused introduces a potential single point of failure that may jeopardize the reliability of all compositions in which it participates if an unexpected error condition occurs. |
| **Solution** | Reusable services can be deployed via redundant implementations or with failover support. |
| **Application** | The same service implementation is redundantly deployed or supported by infrastructure with redundancy features. |
| **Impacts** | Extra governance effort is required to keep all redundant implementations in synch. |
| **Principles** | Service Autonomy |
| **Architecture** | Service |

**Table 12.2**
Profile summary for the Redundant Implementation pattern.

### Problem

Agnostic services are prone to repeated reuse by different service compositions. As a result, each agnostic service can introduce a single point of failure for each composition. Considering the emphasis on repeated reuse within service-orientation, it is easily foreseeable for every complex composition to be comprised of multiple agnostic services that introduce multiple potential points of failure (Figure 12.10).

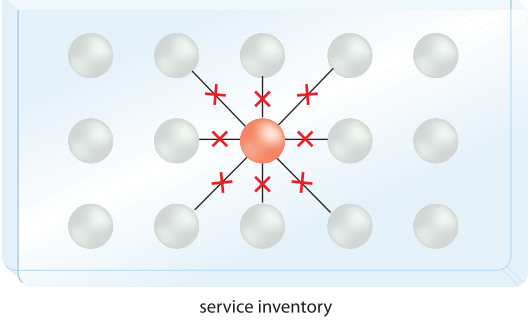



**Figure 12.10**
When a highly reused service becomes unexpectedly unavailable, it will jeopardize all of its service consumers.

## Solution

Multiple implementations of services with high reuse potential or providing critical functionality can be deployed to guarantee high availability and increased reliability, even when unexpected exceptions or outages occur (Figure 12.11).

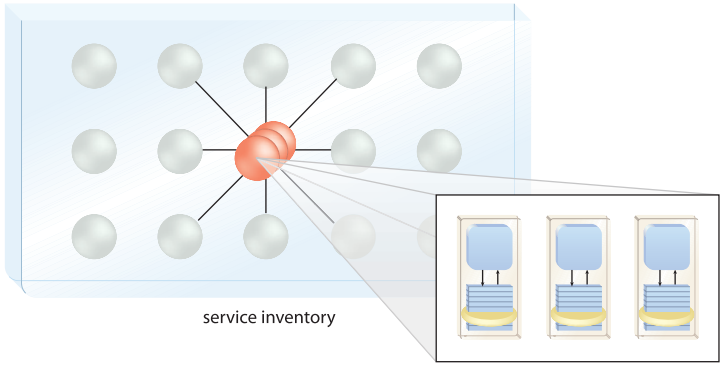



**Figure 12.11**
Having redundant implementations of agnostic services provides fail-over protection should any one implementation go down.

## Application

When services are actually redundantly deployed, there are several ways in which this pattern can be applied:

- Different redundant service implementations can be established for different sets of service consumers.
- One service implementation is designated as the official contact point for consumers, but it is further supported by one or more backup implementations that are used in case of failure or unavailability.

Figure 12.12 illustrates the first variation where the same service is deployed twice; once for access by internal service consumers and again for use by external consumers. This scenario also highlights how this pattern can be applied to various extents. For example, The core service logic may be exactly duplicated in both implementations, but the contracts may, in fact, be different to accommodate the different consumer types, as per Concurrent Contracts (421).

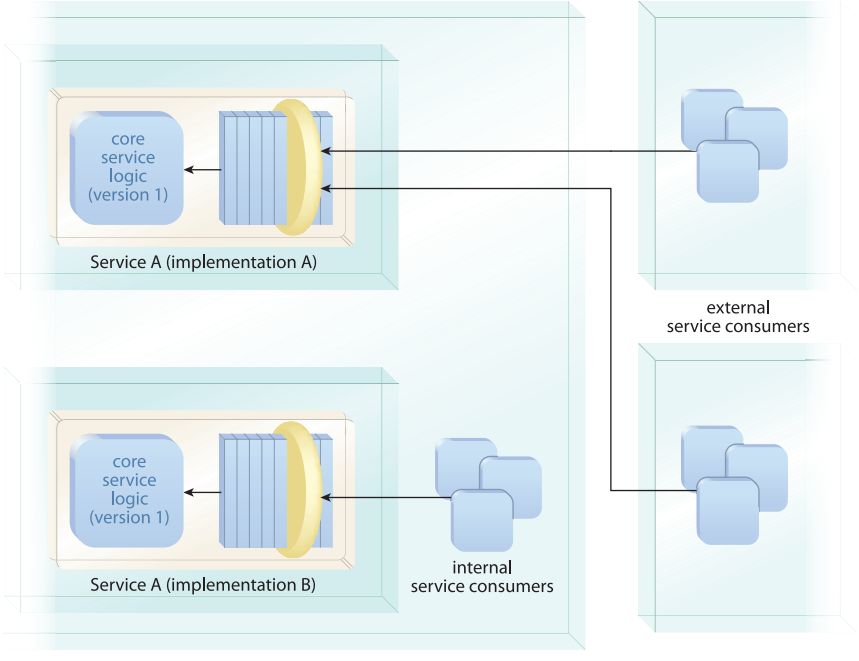



**Figure 12.12**
Service A has multiple service contracts as well as a redundant implementation, allowing this service to facilitate a wide range of consumer programs.

## Impacts

While the application of Redundant Implementation will improve the autonomy, reliability, and scalability of services and the service inventory as a whole, it clearly brings with it some tangible impacts, the foremost of which are increased infrastructure requirements and associated, operational-related governance demands.

For example, additional hardware and administration effort may be needed for each redundantly implemented service and additional governance is required to further keep all duplicated service architectures in synch to whatever extent necessary.

## Relationships

Agnostic services naturally have the most concurrent usage demands and therefore have the greatest need for this pattern, which is why it is important for services defined via Entity Abstraction (175) and Utility Abstraction (168). However, even non-agnostic services, such as those realized via Inventory Endpoint (260) may require Redundant Implementation due to reliability demands.

Composition Autonomy (616) will often repeatedly apply Redundant Implementation to ensure that services participating in the composition can achieve increased levels of autonomy and isolation.

Furthermore, establishing a redundant deployment of a service that requires access to shared data sources will usually demand the involvement of Service Data Replication (350).

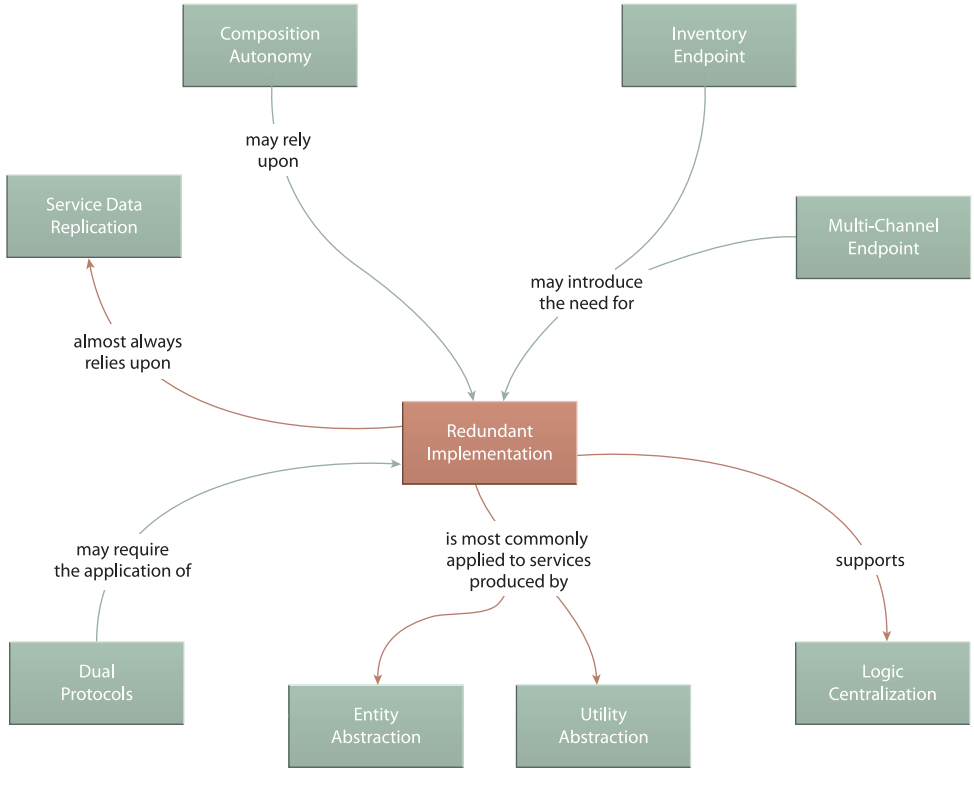



**Figure 12.13**

Redundant Implementation's support for the Service Autonomy design principle affects several other more specialized (autonomy-related) patterns.

**CASE STUDY EXAMPLE**

As illustrated in the case study example for Cross-Domain Utility Layer (267), the Alleywood and Tri-Fold service inventories have been architected to share a set of common utility services.

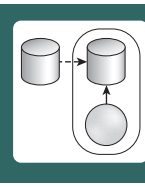
Subsequent to implementing this new cross-domain architecture, some of these utility services naturally became very popular. The Alert service in particular was hit with a consistently high amount of concurrent usage throughout any given work day. Being the service responsible for issuing important notifications when specific pre-defined exception conditions occurred (including policy and security violations), the Alert service was classified as a mission critical part of the overall enterprise architecture.

As a result, a firm requirement was issued, disallowing the Alert service from ever reaching its usage threshold and further requiring chances of service failure be minimized.

To accommodate these requirements, three redundant implementations of the Alert service were created, resulting in four total service implementations. Two were deployed within each environment (Alleywood and Tri-Fold), the second in each environment considered the backup to the first. Intelligent routing agents performed load balancing and failover across each pair of Alert services, as required.

## Service Data Replication

*How can service autonomy be preserved when services require access to shared data sources?*



| | |
|---|---|
| **Problem** | Service logic can be deployed in isolation to increase service autonomy, but services continue to lose autonomy when requiring access to shared data sources. |
| **Solution** | Services can have their own dedicated databases with replication to shared data sources. |
| **Application** | An additional database needs to be provided for the service and one or more replication channels need to be enabled between it and the shared data sources. |
| **Impacts** | This pattern results in additional infrastructure cost and demands, and an excess of replication channels can be difficult to manage. |
| **Principles** | Service Autonomy |
| **Architecture** | Inventory, Service |

**Table 12.3**
Profile summary for the Service Data Replication pattern.

### Problem

Various steps can be taken to increase the overall autonomy and behavioral predictability of services. The components that underlie custom-developed services, for example, can be isolated from other programs into their own process space or even onto dedicated servers. These are relatively straightforward measures because the components, the service contract, and even the extra hardware that may be required are all new to the environment.

However, what usually stands in the way of achieving high levels of autonomy is the fact that even the most isolated service will likely still need to interact with some central database in order to access or even update business data. These repositories are usually shared not just with other services, but with various parts of the enterprise, including the legacy applications they may have been originally built for (Figure 12.14).

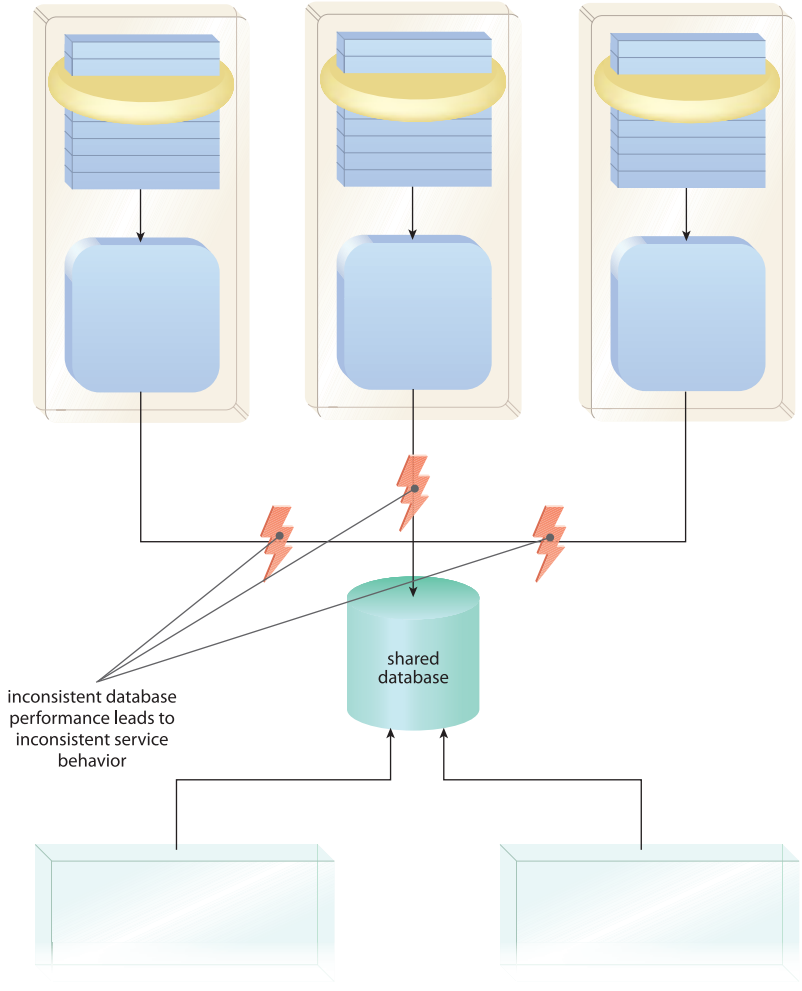



**Figure 12.14**
Multiple services accessing the same shared database will likely encounter locking and performance constraints that will inhibit their individual autonomy.

Although an organization could choose to rebuild their existing data architecture in support of a new service inventory, the cost, effort, and potential disruption of doing so may be prohibitive.

## Solution

Service implementations can be equipped with dedicated databases, but instead of creating dedicated data stores, the databases provide replicated data from a central data source. This way, services can access centralized data with increased autonomy while not requiring exclusive ownership over the data (Figure 12.15).

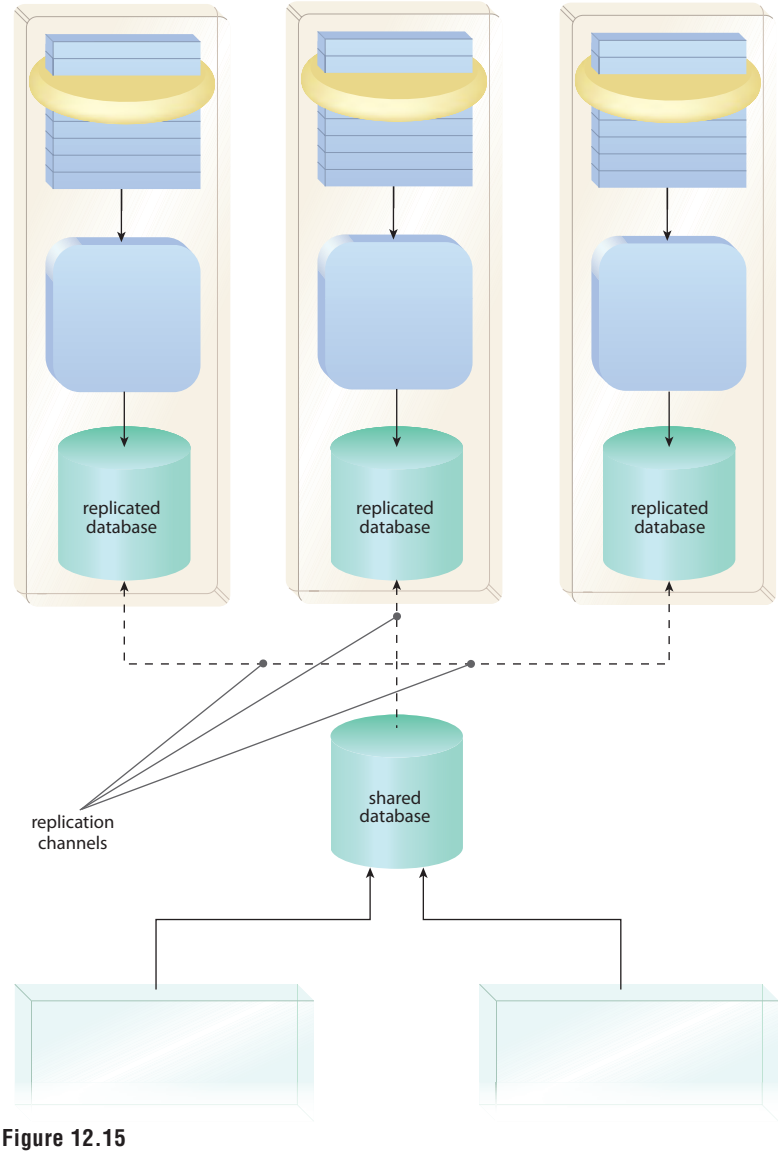



**Figure 12.15**

By providing each service its own replicated database, autonomy is increased and the strain on the shared central database is also reduced.

## Application

This design pattern is especially relevant to agnostic services that are heavily reused and need to facilitate multiple compositions. When this pattern is applied to a large amount of services within a given inventory, it can dramatically reshape the underlying infrastructure of an enterprise's data environment.

Sophisticated data replication architectures may need to be created, and additional design techniques may need to be applied to the databases themselves in order to avoid bottlenecks that can result from an excess of concurrent access and locking. Some replication strategies can even introduce the need for additional satellite databases that provide fully replicated data sets on behalf of a central database but become the contact point for service databases requiring only a subset of the replicated information.

Some services (especially those providing reporting-related capabilities) may only require read access to data, which can be fulfilled by a one-way data replication channel. Most services, though, end up requiring both read and update abilities, which leads to the need for two-way replication.

Furthermore, modern replication technology allows for the runtime transformation of database schemas. As long as the performance and reliability is acceptable, this feature can potentially enable the replicated database to be tuned for individual service architectures.

## Impacts

As stated earlier, repeated application of this design pattern can result in costly extensions to the infrastructure in order to support the required data replication in addition to costs associated with all of the additional licenses required for the dedicated service databases. Furthermore, in order to support numerous two-way data replication channels, an enterprise may need to implement a sophisticated and complex data replication architecture, which may require the need to introduce additional, intermediate databases.

## Relationships

Service Data Replication is a key pattern applied in support of realizing the Service Autonomy design principle. It ties directly into the application of Redundant Implementation (345) and Composition Autonomy (616) because both aim to reduce service access to shared resources and increase service isolation levels. To access or manage replicated data may further involve some form of legacy interface, as per Legacy Wrapper (441).

Data replication can also play a role in service versioning and decomposition. As shown in Figure 12.11, a replicated data source may be required to support isolated capability logic as defined via Distributed Capability (510), or it may be needed to support already decomposed capability logic resulting from Proxy Capability (497).

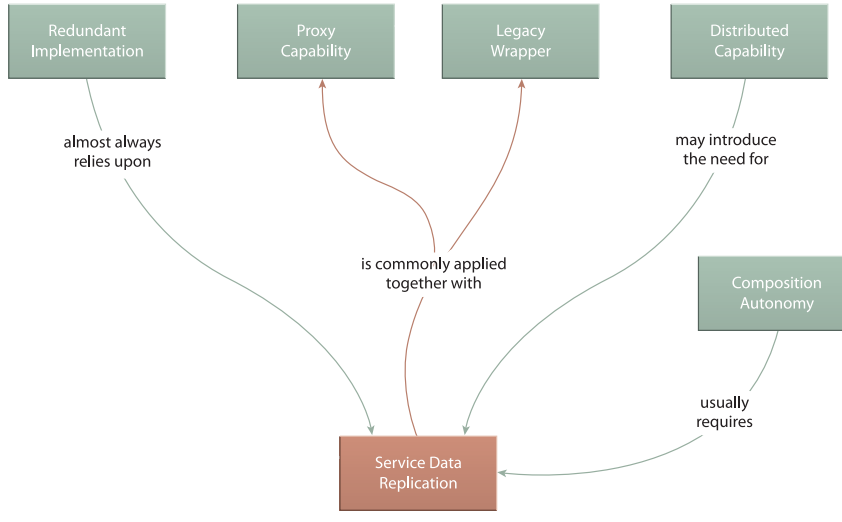



**Figure 12.16**
Service Data Replication helps reduce the requirements for shared data access and therefore supports a series of autonomy-related patterns.

## CASE STUDY EXAMPLE

The FRC Assessment Reports service is responsible for generating and dispensing historical reports for specific registered companies. These reports are used to evaluate and determine annual fines and registration fees (which increase based on the number of violations).

To carry out its reporting functions, this service is required to query the following four databases:

- Registrant Contact (primarily provides company profile information)
- Registrant Activity (includes all incidents, violations, appeals, payments, and other types of historical data)
- Assessment Fees (contains past and current fee schedules and related information)
- Assessment Activity (consists entirely of historical assessments data)

Based on existing design standards that enforce Logic Centralization (136), the first three repositories need to be accessed via the Registrant and Assessment services.

Except for the Assessment Fees database, all of the repositories are used by other legacy applications within the FRC enterprise and now also by other services. Therefore, report generation times can fluctuate, depending on how much shared access is occurring when the Reports service is issuing its queries.

Recently, a new business requirement came about whereby field agents for the FRC would be able to perform assessments on-site while visiting and meeting with registered companies. To perform this task remotely introduced the need for field staff to use portable tablet devices capable of issuing the queries.

To accommodate remote access (especially in regions with limited connectivity) and the increased usage imposed by the new field agent user group, it was decided to improve the response times of assessment report generation by establishing dedicated databases for the Registrant and Assessment Reports services (Figure 12.12).

These databases would be entirely comprised of data replicated from the Registrant Contact, Registrant Activity, and Assessment Activity repositories. Only the subset of data actually required for the reports was replicated and refreshed on a regular basis. The result was a significant increase in autonomy for both Registrant and Assessment Reports services, allowing report generation to be delivered more consistently.

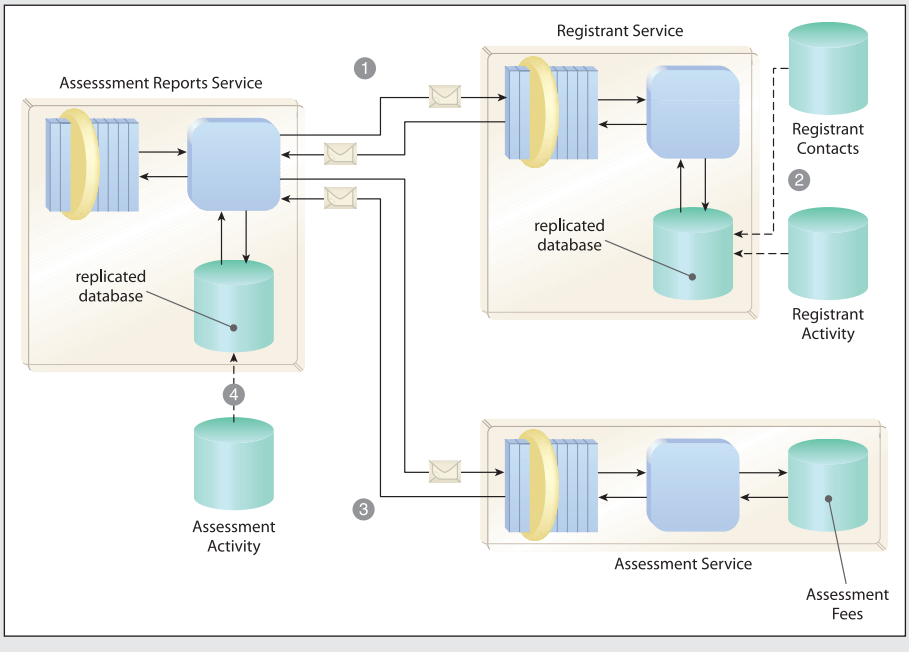
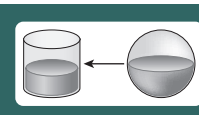



**Figure 12.17**

The Assessment Reports service first invokes the Registrant service to request Registrant profile and activity data (1). The Registrant Service retrieves this data via a dedicated database comprised of data replicated from the Registrant Contact and Registrant Activity repositories (2). Next, the Assessment Reports service requests assessment fee data via the Assessment service (3). This service already has a dedicated database that does not require replication. Finally, the Assessment Reports service retrieves data from its own replicated Assessment Activity database (4).

## Partial State Deferral

*How can services be designed to optimize resource consumption while still remaining stateful?*



| | |
|---|---|
| **Problem** | Service capabilities may be required to store and manage large amounts of state data, resulting in increased memory consumption and reduced scalability. |
| **Solution** | Even when services are required to remain stateful, a subset of their state data can be temporarily deferred. |
| **Application** | Various state management deferral options exist, depending on the surrounding architecture. |
| **Impacts** | Partial state management deferral can add to design complexity and bind a service to the architecture. |
| **Principles** | Service Statelessness |
| **Architecture** | Inventory, Service |

**Table 12.4**
Profile summary for the Partial State Deferral pattern.

### Problem

When services are composed as part of larger runtime activities, there is often a firm need for the service to remain active and stateful while other parts of the activity are being completed.

If the service is required to hold larger amounts of state data, the state management requirements can result in a significant performance drain on the underlying implementation environment. This can be wasteful when only a subset of the data is actually required for the service to accommodate the activity.

In high concurrency scenarios environments, the actual availability of the service can be compromised where accumulated, wasted resources compound to exceed system thresholds (Figure 12.18).

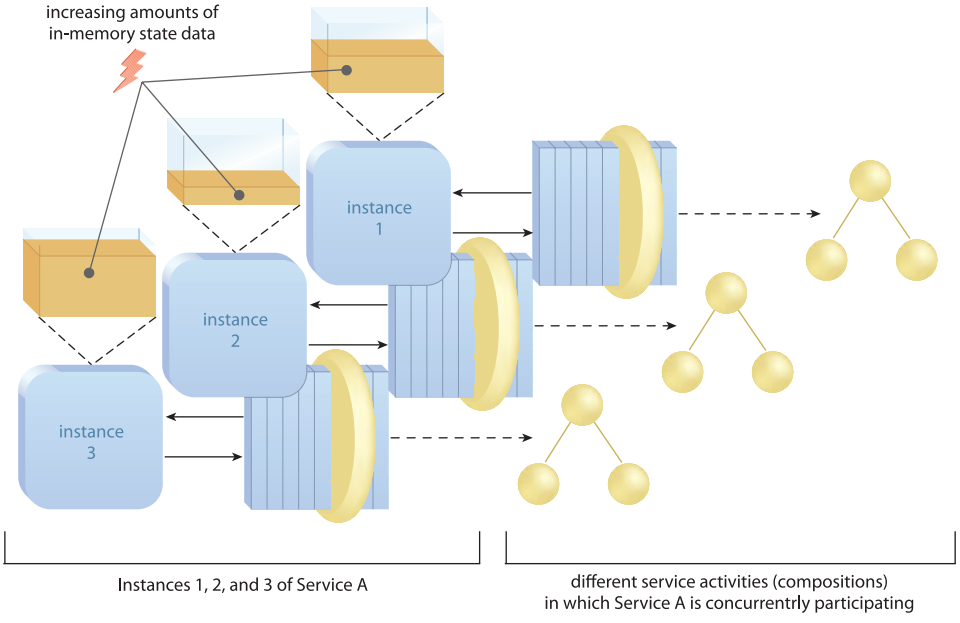



**Figure 12.18**

In concurrent usage scenarios, stateful services will require that multiple service instances be invoked, each with its own measure of state-related memory consumption requirements.

## Solution

The service logic can be designed to defer a *subset* of its state information and management responsibilities to another part of the enterprise. This allows the service to remain stateful while consuming less system resources (Figure 12.19). The deferred state data can be retrieved when required.

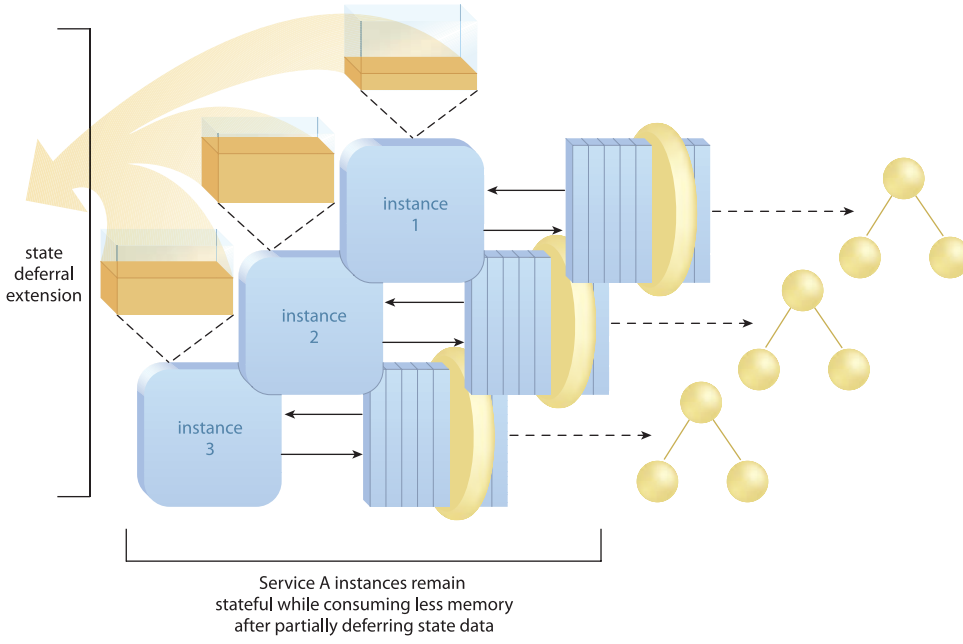



**Figure 12.19**
Applying this pattern results in the same amount of concurrent service instances but less overall state-related memory consumption.

## Application

This design pattern is almost always applied for the deferral of large amounts of business state data, such as record sets or code lists. The general idea is for these bodies of data to be temporarily off-loaded. To accomplish this, an effective state delegation option is required. This may preclude the use of State Repository (242) unless virtual databases can be utilized to make the writing and retrieval of data efficient and responsive.

Partial State Deferral can be effectively used in conjunction with Stateful Services (248) or State Messaging (557) so that state data transmissions can occur without writing to disk. Any state deferral extension can be used in support of this pattern, as long as the performance hit of transferring state data does not introduce unreasonable lag time to the overall activity so that the extension does not undermine the performance gain sought by the pattern itself.

Services designed with this pattern can be further optimized to minimize lag time by retrieving deferred state data in advance.

**NOTE**

For descriptions of different types of state data and levels of service statelessness, see SOAGlossary.com.

## Impacts

Most state management deferral options require that the service move and then later retrieve the state data from outside of its boundary. This can challenge the preference to keep the service as a self-contained part of an inventory and can also bind its implementation to the technology architecture. The resulting architectural dependency may result in governance challenges should standard state management extensions ever need to be changed.

Furthermore, the routines required to program service logic that carries out runtime state data deferral and retrieval add design and development complexity and effort. Finally, if the aforementioned optimization is not possible, the retrieval of large amounts of business data as part of a sequential processing routine will introduce some extent of lag time.

**NOTE**

The target state sought by this design pattern corresponds to the *Partially Deferred Memory* statelessness level described in Chapter 11 of *SOA Principles of Service Design*.

## Relationships

This specialized pattern has relationships with the other state management-related patterns, namely State Repository (242), Service Grid (254), State Messaging (557), and Stateful Services (248), and also provides a common feature used in orchestration environments, as per Process Centralization (193). The application of Canonical Resources (237) can further affect how this pattern is applied.

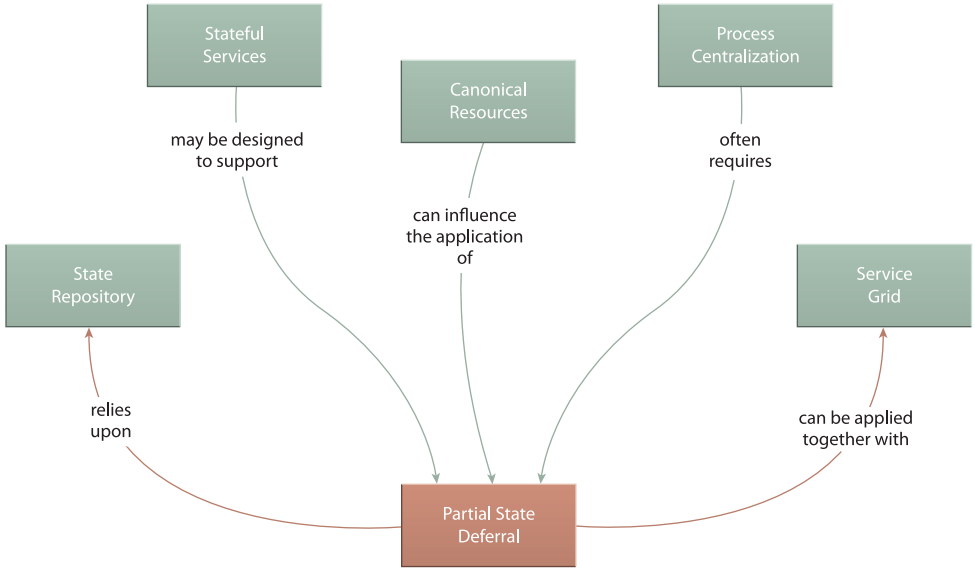



**Figure 12.20**
Partial State Deferral has basic relationships with other patterns that support or benefit from state management delegation.

## CASE STUDY EXAMPLE

The FRC Area Policy Report service (also described in the Composition Autonomy (616) case study example section) is required to access the Area service and then the Policy Checks service, the latter of which is located on a remote server. Because policy data does not change on a frequent basis and because on any given day most queries issued by the Area Policy Report service are generally related to the same areas, an opportunity is discovered to optimize the composition architecture by applying Partial State Deferral.

Essentially, whenever one or more instances of the Area Policy Report task service are active, the retrieved policy data is stored in a local state repository. Because during the course of a normal working day the majority of reports relate to the same group of areas, the stored policy data is useful to most instances of this service.

As shown in Figure 12.21, instances of the Area Policy Report task service remain stateful but are not required to explicitly retrieve or store the policy data.

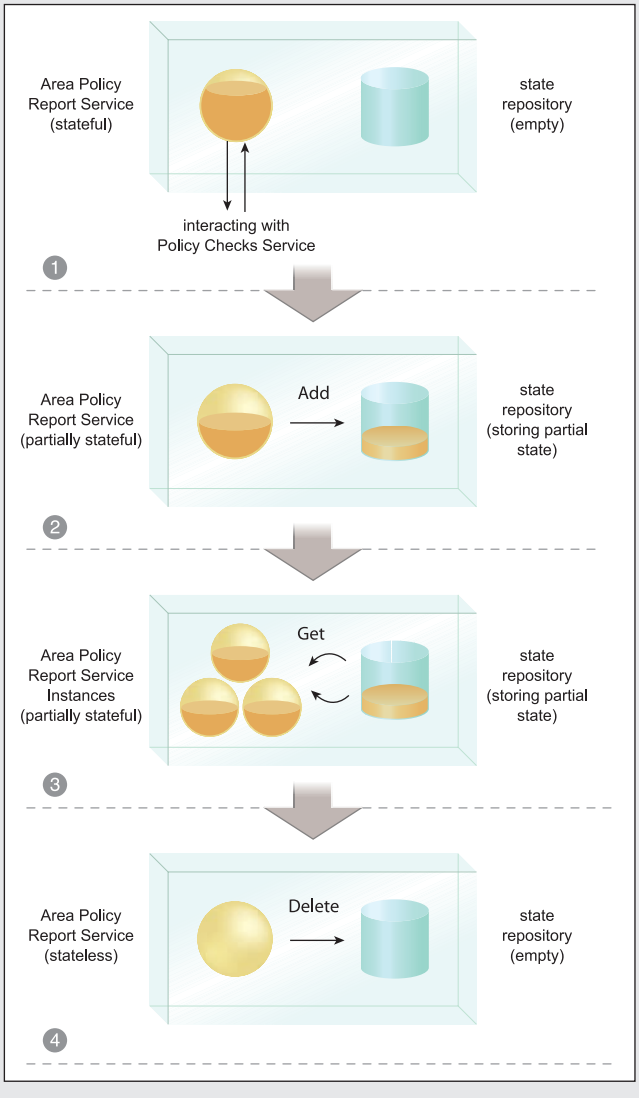


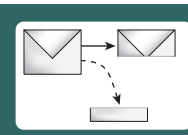

**Figure 12.21**

The first instance of the Area Policy Report service is responsible for retrieving the policy data from the Policy Checks service (1) and populating the state repository (2). Subsequent instances are free to query the state repository (3) instead of accessing the Policy Checks service. Finally, the last instance is responsible for clearing the state repository (4).

# Partial Validation

By David Orchard, Chris Riley

*How can unnecessary data validation be avoided?*



| | |
|---|---|
| **Problem** | The generic capabilities provided by agnostic services sometimes result in service contracts that impose unnecessary data and validation upon consumer programs. |
| **Solution** | A consumer program can be designed to only validate the relevant subset of the data and ignore the remainder. |
| **Application** | The application of this pattern is specific to the technology used for the consumer implementation. For example, with Web services, XPath can be used to filter out unnecessary data prior to validation. |
| **Impacts** | Extra design-time effort is required and the additional runtime data filtering-related logic can reduce the processing gains of avoiding unnecessary validation. |
| **Principles** | Standardized Service Contract, Service Loose Coupling |
| **Architecture** | Composition |

**Table 12.5**
Profile summary for the Partial Validation pattern.

## Problem

Agnostic services are designed with high reuse potential in mind, and therefore there is a constant emphasis on providing generic capabilities that can accommodate a wide range of possible consumers.

Although this approach leads to increased reuse opportunities, it can also impose unreasonable validation requirements upon some consumers. A typical example is when a capability is designed to be intentionally coarse-grained in order to provide a broad data set in its response messages. The set of data may only be useful to a subset of the service consumers the remaining of which will be forced to validate the message data upon receiving it but then discard data that is not relevant to their needs (Figure 12.22).

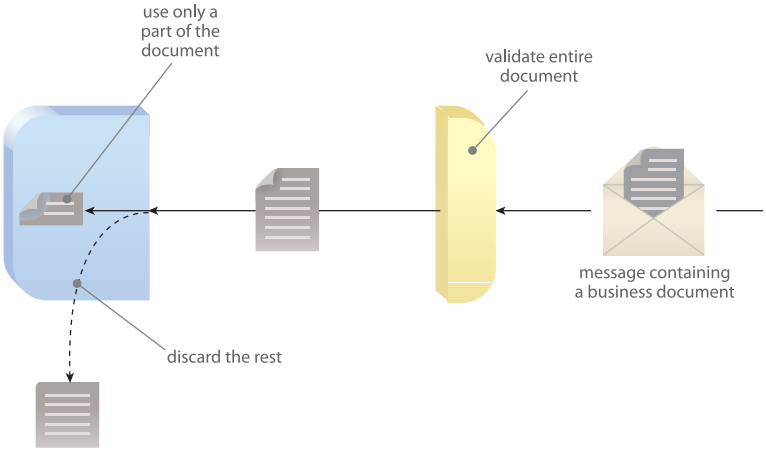



**Figure 12.22**
When a service consumer requires only a subset of the data provided to it by the agnostic service, it is expected to validate the entire data set (message payload) before discarding the unnecessary message data.

## Solution

The service consumer is intentionally designed to not fully comply to the service contract. Instead, its validation logic is tuned to only look for and validate message data relevant to its needs, thereby ignoring the rest (Figure 12.23). This reduces consumer processing requirements and decreases the extent to which some consumers need to couple themselves to the service contract.

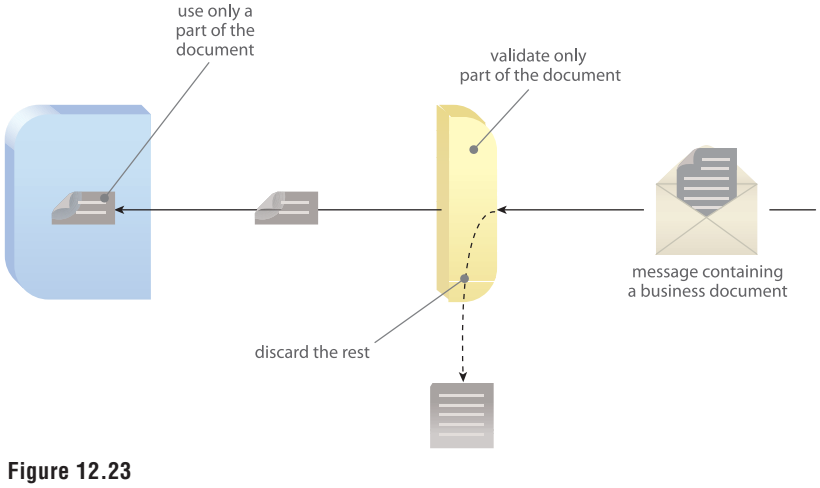



**Figure 12.23**
Because the irrelevant data is ignored prior to validation, it is discarded earlier and avoids imposing unnecessary validation-related processing upon the consumer.

## Application

Partial Validation is applied within the consumer program implementation. Custom routines are added to allow for the regular receipt and parsing of incoming service messages, while then avoiding actual validation of irrelevant data.

A typical algorthim used by these routines would be as follows:

1. Receive response message from service.
2. Identify the parts of the message that are relevant to the consumer's processing requirements.
3. Validate the parts identified in Step 2 and discard the balance of the message contents.
4. If valid, retain the parts identified in Step 2. Otherwise, reject the message.

Partial Validation routines can be located within the core consumer business logic or they can be abstracted into an event-driven intermediary, as per Service Agent (543). When services assume the consumer role by composing other services, this type of logic may also be suitable for abstraction via Service Façade (333).

## Impacts

The custom programming required by this pattern can add to the design complexity of the overall consumer program logic. Furthermore, the extra processing required by the consumer to look for and extract only relevant data can impose its own processing overhead.

## Relationships

Depending on how Partial Validation is applied, it may make sense to combine it with Service Agent (543) or Service Façade (333). Although not directly related to the application of Validation Abstraction (429), these two patterns do share common goals.

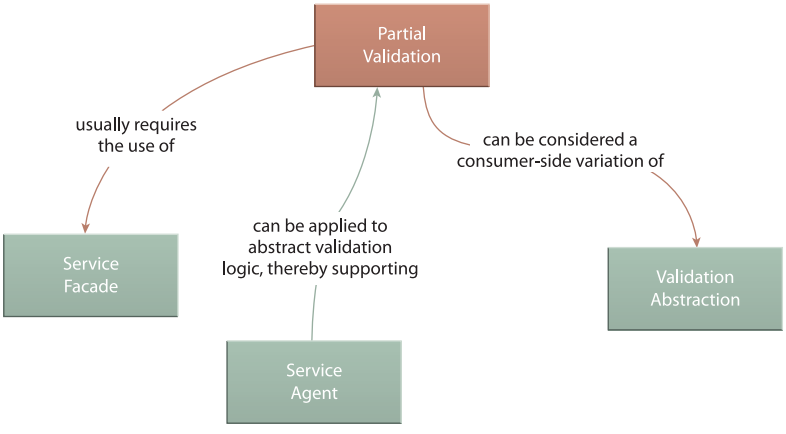



**Figure 12.24**
Partial Validation introduces internal, consumer-side processing logic and therefore has limited relationships with other patterns.

## CASE STUDY EXAMPLE

The FRC Assessment Reports service (described earlier in the Service Data Replication (350) case study example) is required to access the Registrant service in order to request registrant profile data for one of its reports (see Figure 12.12). These reports are often parameter-driven, meaning that they can vary in scope and content depending on the reporting parameters provided to the Assessment Reports service by a given consumer.

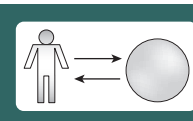
Because of this variance in report content and because the Assessment Reports service is always required to invoke the same Get operation that returns entire profile documents, it often ends up with more registrant data than it actually needs. Its original design simply accepted and validated incoming messages and then made the entire message contents available to the report generation routines. However, as the amount of concurrent usage increases, so does the complexity of some reports, leading to increased resource requirements for this service.

Due to the enterprise-wide initiative to reduce infrastructure costs, architects do not receive funding to apply Redundant Implementation (345) in order to establish a second implementation of this service for load balancing purposes. This forces them to revisit the service design in order to investigate optimization opportunities.

They soon discover that the internal service logic can be refactored by applying Partial Validation. This enables the service to continue performing its report generation logic while decreasing its processing and memory consumption due to a dramatic reduction in runtime message validation.

# UI Mediator

By Clemens Utschig-Utschig, Berthold Maier,
Bernd Trops, Hajo Normann, Torsten Winterberg



*How can a service-oriented solution provide a consistent, interactive user experience?*

| | |
|---|---|
| **Problem** | Because the behavior of individual services can vary depending on their design, runtime usage, and the workload required to carry out a given capability, the consistency with which a service-oriented solution can respond to requests originating from a user-interface can fluctuate, leading to a poor user experience. |
| **Solution** | Establish mediator logic solely responsible for ensuring timely interaction and feedback with user-interfaces and presentation logic. |
| **Application** | A utility mediator service or service agent is positioned as the initial recipient of messages originating from the user-interface. This mediation logic responds in a timely and consistent manner regardless of the behavior of the underling solution. |
| **Impacts** | The mediator logic establishes an additional layer of processing that can add to the required runtime processing. |
| **Principles** | Service Loose Coupling |
| **Architecture** | Composition |

**Table 12.6**
Profile summary for the UI Mediator pattern.

## Problem

Service-oriented solutions are commonly designed as service compositions that may be comprised of services with varying runtime behaviors and processing demands. When the process being automated by the solution is driven by human interaction via user-interfaces, the quality of the user experience can vary due to these behavioral and environmental irregularities.

Whereas a human user expects immediate responses to requests issued via the user-interface, the underlying services may be not be designed or able to provide these responses in a synchronous and timely manner (Figure 12.25). Poor or inconsistent user experience can lead to a decrease in the usage and overall success of the solution.

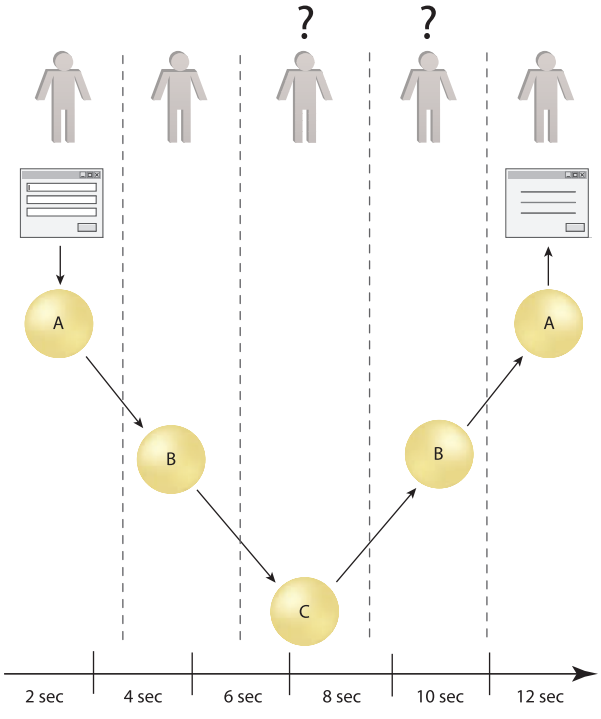



**Figure 12.25**

While services A, B, and C require several seconds to automate a task initiated via a user-interface, the human user receives no indication as to the progress of the task and is left waiting until a result is finally displayed.

## Solution

A mediator service is positioned between a service or service composition and the front-end solution user-interfaces. It is responsible for providing the user with continuous feedback and for gracefully facilitating various runtime conditions so that the underlying processing of the services does not affect the quality of the user experience (Figure 12.26).

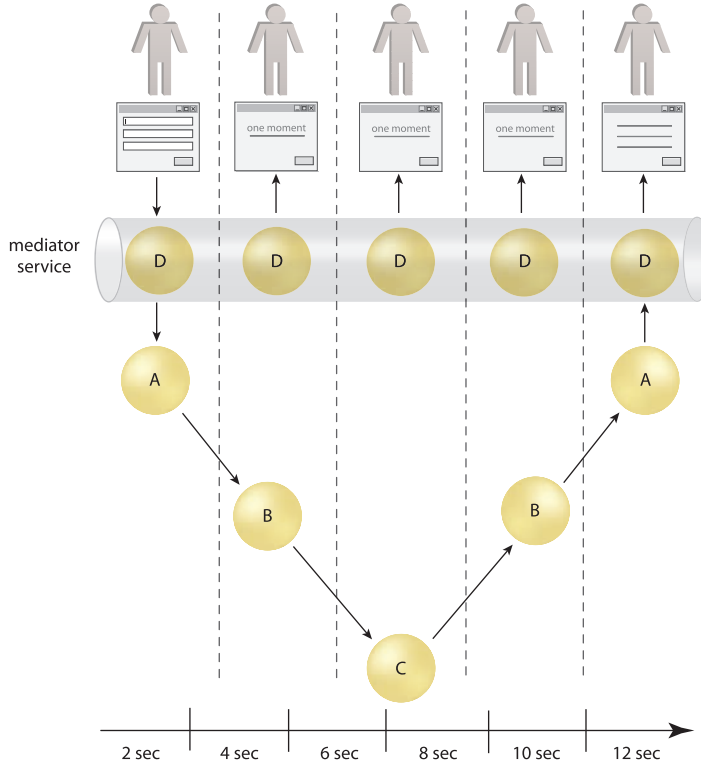



**Figure 12.26**

The mediator service (D) regularly updates the user interface while services A, B, and C work behind-the-scenes to complete the task.

## Application

There are two common methods of applying this pattern:

- Build a mediator service with its own service contract.
- Build a mediator service agent.

The first approach requires that a mediator utility service be created and that the user-interface be designed to bind solely with this service for the duration of a specific task. The mediator service exposes a generic contract with weakly typed capabilities that simply relay request and response messages between the user-interface and underlying service(s). This type of mediator service will contain logic that determines how and when to interact with the user-interface independently.

The second approach requires that this pattern be applied together with Service Agent (543). When locating mediator logic within event-driven agents, request and response messages between the user-interface and services are transparently intercepted, triggering events that kick-off the mediation logic. Agents must be designed to remain stateful during the completion of the task so that they can interact with the user-interface as required.

Common user-interface mediation routines include:

- displaying forms or pages with a progress or status indicator while services are processing a given request
- displaying forms that request additional data from the user
- routing a user task to the next step independently from underlying service processing
- simulating synchronous human-to-solution exchanges while underlying service activities are carried out asynchronously
- gracefully responding to exception or time-out conditions

The mediator essentially preserves a constant correlation between a user session and the process being automated by services. For this purpose, the mediator service or agent may even maintain a correlation ID that is assigned to all incoming and outgoing messages. However, in order for mediation logic to remain agnostic, it will generally not contain any business process-specific rules or logic. Its capabilities are limited to generic interaction routines.

### Impacts

When delivering the mediator logic as a service or a service agent, additional runtime processing is added to the automation of the overall business task due to the insertion of the mediator service layer within the overall composition (and also due to the frequent interaction carried out independently by the mediator logic).

Furthermore, when the mediator exists as a service with its own published contract, user-interfaces are required to bind directly to and interact with the mediator service during the span of an entire business task. This naturally decouples the user-interfaces from the underlying service composition, which can be advantageous if the business logic is subject to change. However, if the mediation logic is agnostic and positioned as part of a reusable utility service, the parent composition logic may actually be responsible for controlling its involvement, leading to a tighter coupling.

## Relationships

Because it represents a form of utility logic, UI Mediator is based on Utility Abstraction (168). Service Agent (543) simply provides an optional implementation medium for this pattern.

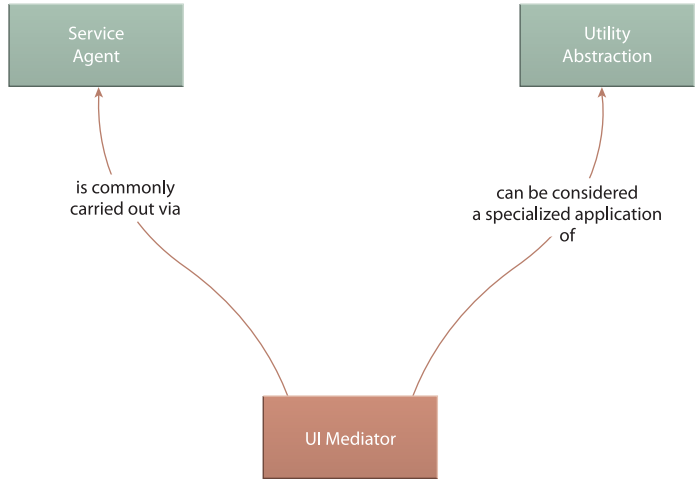



**Figure 12.27**
As a specialized utility service, UI Mediator has few relationships.

### CASE STUDY EXAMPLE

Prior to the purchase of Alleywood Lumber, McPherson had been carrying out an on-going BPM initiative to help automate existing manual processes and to optimize outdated automated processes. With the assimilation of Alleywood operations, McPherson analysts subject a series of former Alleywood business processes to their established business modeling practices.

The first process relates to the online commercial and retail sale of lumber. Alleywood has a simple Web site in place that allows clients to perform the following tasks sequentially:

1. Initiate a search on the availability of different types of lumber.
2. Assemble an order from the available items.
3. Indicate the shipping details.

The Web site contains scripts that run on the Web server and integrate with an outdated legacy database in which lumber items are stored. Because this database is shared throughout the Alleywood enterprise, its performance and response time can vary dramatically. This primarily affects the time it takes to complete Step 1.

Analysts investigate the history of this commerce site by studying usage logs. After consolidating some of the statistics using a special tool, they discover that the search action can take up to 60 seconds to complete. They further find out that 30% of users who initiate a search abandon the Web site when the query time exceeds 20 seconds, whereas when the query time is less than 5 seconds, 90% of searches result in actual orders.

These metrics are compiled into a report that provides a series of recommendations as to how the online lumber ordering site can be improved, with an emphasis on user experience. This report is passed on to the architecture team, which responds by making a number of changes:

- Service Data Replication (350) is applied to establish a dedicated database for the site.
- A Retail Lumber service is created to provide standardized data access to the database.
- UI Mediator is applied using Service Agent (543) to ensure that long query times or unexpected erratic behavior do not affect the user experience.

The mediator logic contains the following built-in rules:

- If there is no response from the Retail Lumber service within 5 seconds, display a progress indicator page in the user's browser.
- If there is no response from the Retail Lumber service after 15 seconds, display a Web page with a message explaining that the system is currently not available but that the requested information will be e-mailed to the user shortly (the e-mail address is captured as part of the login credentials required to access the site).
- If the Retail Lumber service responds with no data, display the original form along with a message indicating that different search parameters must be provided.

The mediator agent helps establish an improved user experience that appears to be synchronous and interactive, regardless of the behavior of the Retail Lumber service and its underlying database.


# MASHUP PATTERNS

## Designs and Examples for the Modern Enterprise

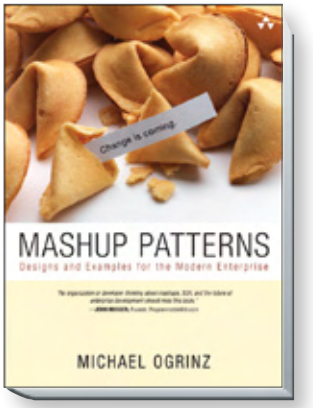
*"No organization or developer thinking about mashups, SOA, and the future of enterprise development should miss this book."*

—**JOHN MUSSER,** Founder, ProgrammableWeb.com

MICHAEL OGRINZ

Michael Ogrinz

BUY ME




# Mashup Patterns

## Designs and Examples for the Modern Enterprise

### Indispensable Patterns and Insights for Making Mashups Work in Production Environments

Using new mashup tools and technologies, enterprise developers can impose their own APIs on everything from Web sites and RSS feeds to Excel and PDF files–transforming the Web into their own private information source. In *Mashup Patterns*, Michael Ogrinz systematically reveals the right ways to build enterprise mashups and provides useful insights to help development organizations avoid the mistakes that cause mashups to fail.

Drawing on extensive experience building business-critical mashups, Ogrinz offers patterns and realistic guidance for every stage of the mashup development lifecycle and virtually every key issue developers, architects, and managers will face. Each pattern is documented with a practical description, specific use cases, and crucial insights into the stability of mashups built with it. Ogrinz concludes by presenting six start-to-finish case studies demonstrating mashup patterns at work in actual enterprise projects.

COVERAGE INCLUDES:

- Understanding the relationships among mashups, portals, SOA, EAI/EII, and Software as a Service
- Implementing core activities such as data management, surveillance, clipping, transformation, enrichment, publication, and promotion
- Optimizing security, privacy, accessibility, usability, and performance
- Managing mashup development, from planning and governance through integration, testing, and deployment
- Enhancing basic mashups with search, language translation, workflow support, and other improvements
- Performing effective load and regression testing
- Avoiding “anti-patterns” that cause enterprise mashups to fail

AVAILABLE

- BOOK: 9780321579478
- SAFARI ONLINE Safari Books Online
- EBOOK: 0321579836
- KINDLE: COMING SOON

## About the Author

MICHAEL OGRINZ is a principal architect at one of the world’s largest financial institutions. His business focus is to identify and integrate emerging technologies into the enterprise and to create innovative and competitive solutions. A frequent industry speaker on various facets of Enterprise 2.0, Michael has been instrumental in enhancing the computing environment at his firm through his work on user interfaces and usability, wikis and blogs, and, most recently, mashups.

Michael cofounded localendar.com, a classic “garage start-up” that demonstrates how the “Long Tail” applies to online calendars as much as anything else. The niche site has provided more than 400,000 schools, churches, and clubs with simple online scheduling services since its inception more than eight years ago.

Addison Wesley

informit.com/aw

# Chapter 1

# Understanding Mashup Patterns

Collaborators welcome![1]

## Introduction

When the World Wide Web was first unveiled, "collaborators" referred to one small segment of the population: nerds.[2] The first browser ran on a computer that almost no one outside of a university or research lab used.[3] The Web itself consisted of a lone site[4] (WWW Growth, Figure 1.1). Yet from this singularity, a new universe would soon emerge.

The amount of content didn't grow much until two years later. That was when the first of several "Big Bangs" would occur. In 1993, the first PC-based program capable of browsing the Web was released.[5] Its introduction instantly put the Web within the reach of a far larger audience. Even so, Internet connectivity remained largely restricted to universities, research institutes, and corporations. Consumers enjoyed online communities, but generally did so via prepackaged, fenced-in services such as Compuserve, Prodigy, and America Online (AOL). Connectivity was achieved through slow "dial-up" connections over telephone lines. Access to content was typically billed at an hourly rate.

1. From Tim Berners-Lee's first public Usenet post announcing the public availability of the first Web server and browser in 1991.
2. A contingent of which I am proud to proclaim myself a member.
3. The NeXT workstation, conceived by computer luminary Steve Jobs.
4. Tim Berners-Lee invented the World Wide Web in 1989 while working at the CERN Particle Physics Laboratory.
5. NCSA Mosaic, released in 1993.

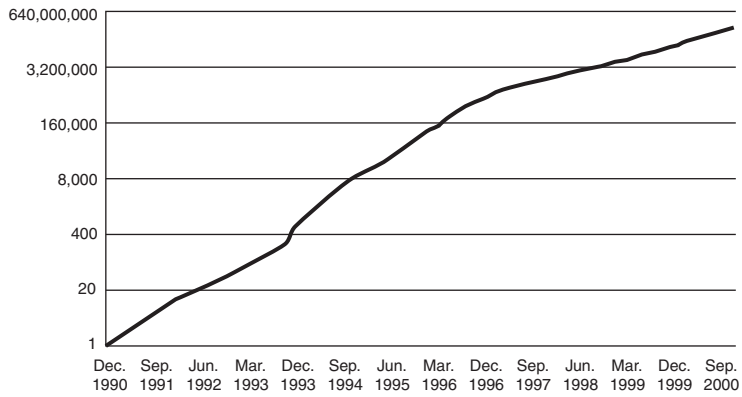



**Figure 1.1** *The growth of the World Wide Web: number of Web sites, 1990–2000*

By 1994, the first independent Internet service providers (ISPs) had begun to pop up. By installing special software on their computers, consumers could access the entire content of the Web (almost 1,000 sites!). AOL began to open up Web access for its millions of subscribers. Prices universally moved to flat-rate monthly charges. WYSIWYG ("What you see is what you get") HTML editors appeared and made creating Web pages just a bit easier. In response, the second explosion in Web growth occurred. By 1996, corporations didn't see a Web presence as a luxury, but rather as a necessity. What better way to instantly push content to the consumer? The Web was viewed as a new media channel that offered endless opportunities for commercial success.

If the waning years of the past century had a motto, it certainly wasn't "Collaborators welcome"; "Venture capital welcome" is probably more accurate. Fueled by ill-conceived business plans and wild speculation, a worldwide expansion of the Web's underlying infrastructure took place. Meanwhile, the browser jumped from home computers to cell phones and mobile devices for the first time. High-speed cable and DSL "broadband" connectivity options became ubiquitous. The third explosion was the popping of the Web bubble, which saw these ventures implode en masse when they failed to turn a profit. This event marked the end of the first wave of the Web's evolution, which in hindsight we label Web 1.0.

## Web 2.0

In the aftermath of the Web 1.0 crash, the glut of infrastructure kept the costs of going online low. That simple fact helped attract even more users to come

online. A few companies began to figure out how to leverage the Web without going bankrupt. Collectively, their embrace of the Internet represented the slow expansion of the Web from that last primordial blast. New marketplaces evolved as sites like eBay linked buyers and sellers from around the globe. These online flea markets, in turn, spawned communities that helped pioneer the concepts behind new social networking sites like MySpace and Facebook.

By 2006, the firms that had simultaneously feared and tried to control Web 1.0 looked up from licking their wounds and saw the dawn of a new paradigm. In a symbolic changing of the guard, "old media" giant *Time* magazine announced the Person of the Year was "You."[6] There was no great single occurrence that made this milestone possible. Rather, the driving force was the confluence of many events: the spread of cheap broadband access, the Web-enabling of multiple devices, the arrival of new communication environments, and the emergence of cooperative environments for organizing information. Collaborators were finally running the show.

Industry figurehead Tim O'Reilly is credited with popularizing the term "Web 2.0" to define this new age:

> Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as platform, and an attempt to understand the rules for success on that new platform.[7]

A simpler working definition is that Web 2.0 is a shift from *transaction-based* Web pages to *interaction-based* ones. This is how the power of "You" is mashed, mixed, and multiplied to create value. Social-networking sites, folksonomies (collaborative tagging, social bookmarking), wikis, blogs, and mashups are just some of the components that make this possible. The success of sites such as Facebook, wikipedia, flikr, and digg has demonstrated that democratization of content creation and manipulation is powering the latest wave of Internet growth.

The underlying driver of Web 2.0 is flexibility. The one trait technologies slapped with the Web 2.0 moniker share is that they are extremely (and perhaps sometimes unintentionally) malleable. The successful products don't break when a user tries to extend them beyond their original design; they bend to accept new uses. Two success stories of the new Web illustrate this principle:

> flickr was started by Caterina Fake and Stewart Butterfield as an add-on feature for a video game they were developing. The idea was to allow players to save and share photos during gameplay. When they realized that bloggers needed a convenient way to store and share photos, Fake and Butterfield started adding blog-friendly features.

6. *Time* magazine, December 13, 2006.

7. http://radar.oreilly.com/archives/2006/12/web-20-compact-definition-tryi.html

Opening up their architecture to allow users of the site to create custom enhancements fueled their viral spread. The original game was ultimately shelved and flickr was sold to Yahoo! a year later for an undisclosed sum.

Deli.cio.us grew from a simple text file that its founder, Joshua Schachter, used to keep track of his personal collection of tens of thousands of Web site links. When the site went public in 2003, it spawned a host of add-ons. The concept of associating data with simple keywords to aid in organization wasn't new, but the cooperative "social tagging" aspect of deli.cio.us resonated with the frustrations of other Internet users.

## Enterprise 2.0

Inevitably, when people discover a useful tool outside the workplace, they want to use it at the office as well. This happened years earlier when employees began sneaking personal computers into their offices to make it easier to manage spreadsheets and documents. More recently, end users have imported instant messaging and unlimited email[8] services from external sources.

User demand for Web 2.0 technologies within existing corporate infrastructure is the catalyst for Enterprise 2.0.[9] The challenge for firms is to integrate these new peer-based collaboration models with legacy technologies and mindsets. Figure 1.2 illustrates three areas that established organizations have typically established to control how solutions are delivered.

Enterprise 2.0 breaks down traditional divisional barriers and encourages building bridges. The managerial structure does not change, but the ability to conceive solutions and access the technology to deliver them is available to everyone (as shown in Figure 1.3).

Changing the social structure of a firm is termed "soft reorganization." Its consequence is movement away from fixed roles and responsibilities and toward a more open and unrestricted workplace. The phrase "economies of scale" refers to the cost advantages associated with large-scale production. We term the benefits of Enterprise 2.0 the "economies of collaboration." How are they established?

8. When Gmail (Google Mail) was announced in April 2004, it offered 1 gigabyte of message storage. This was well beyond the storage limit most corporate mail systems impose on their employees.

9. McAfee, Andrew. "Enterprise 2.0: The Damn of Emergent Collaboration." *Sloan Management Review,* Vol. 47, Spring 2006.

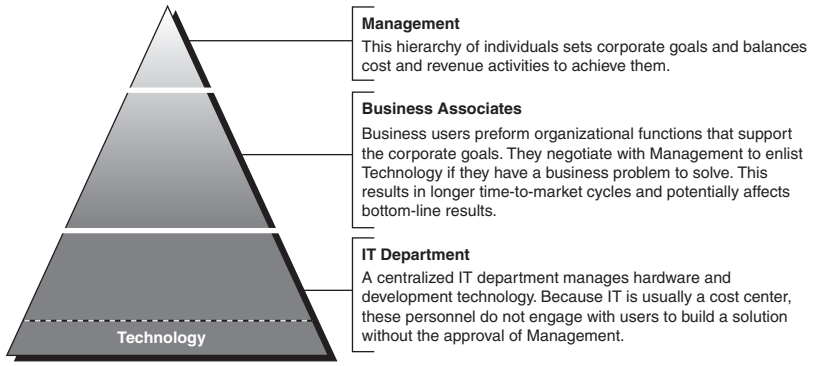



**Figure 1.2** *Typical organizational hierarchy*

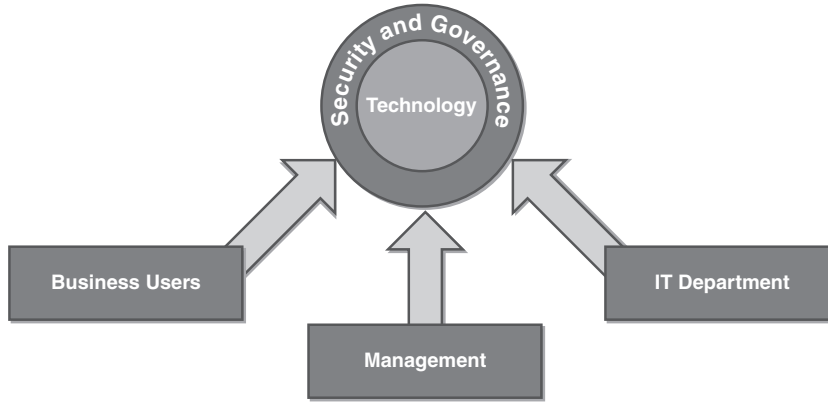



**Figure 1.3** *Traditional barriers to solution delivery are removed in Enterprise 2.0. Each segment of an organization now has equal access to technology. To leverage this new environment, powerful (yet user-friendly) tools are introduced. These tools enable associates outside traditional IT to create their own solutions.*

- Nontechnical users are empowered to create application solutions without engaging management or IT personnel in the process. This agility leads to shorter time-to-market cycles.
- Folksonomies replace strict taxonomies (see the "Folksonomies versus Taxonomies" sidebar). Newly discovered connections between data and processes can be exploited to add business value.

- New communication tools mine "the wisdom of the crowd" to encourage collaboration and innovation, a technique known as crowdsourcing (see the "Crowdsourcing" sidebar).

Open interaction can help teams discover how the other lines of business operate. This knowledge, in turn, leads to changes that strengthen relationships across departments.

- IT must learn more about the business associates' goals, and create an environment that facilitates the rapid construction of products that they require.
- Members of the business team must participate more directly in the engineering process (either on their own or in partnership with IT), which requires some knowledge about development best practices.
- Management needs to cede some control to other teams and should work with all associates to encourage collaboration. This may entail:
  - Funding the necessary infrastructure.
  - Allowing cross-pollination between business teams.
  - Being open to ideas from nontraditional sources.

Security becomes a universal concern as the lines between teams vanish. The former "checks and balances" approach doesn't work when small teams are creating end-to-end solutions. In this collaborative milieu, firms have to strike a balance between technical controls[10] and education to mitigate risk.

### Folksonomies versus Taxonomies

Taxonomies describe the organization of data within a strict hierarchy. In the business world, they are typically artifacts of established corporate structures. The managerial chain of command establishes processes for the composition, categorization, and flow of information. The structure of a rigid taxonomy may be nonintuitive to outsiders and consequently may restrict the sharing of useful information across the firm.

In a folksonomy, the community takes responsibility for collectively classifying and organizing information through a process known as "tagging." Tagging simply entails

10. For example, putting a formal development process with relevant checkpoints and milestones in place.

labeling content with a few relevant keywords that describe the information or the ways in which it can be used. As more reviewers add and refine tags, it becomes easier to locate and navigate large amounts of information. The process of tagging creates a dynamic knowledge base of material that is not constrained by conventional organizational techniques.

### Crowdsourcing

With crowdsourcing, a problem is framed so that it can be tackled by multiple teams or individuals, working either competitively or as a group effort. User-driven mashups can facilitate this type of mass collaboration in the enterprise, thereby resulting in far more resources contributing to solutions besides traditional IT.

A danger of this approach is that a "herd mentality" might develop that stifles creativity. Some degree of oversight can offset this risk, but care must be taken not to discourage participation.

Crowdsourcing success stories include the Ansari X-Prize, which was designed to encourage low-cost space travel, and Wikipedia, which benefits from the combined contributions of thousands of users.

## The Birth of Mashups

You can have it "good," "fast," or "cheap." Pick any two of the three.
—Classic programmer's adage

Quick, easy, and affordable application development has always been a goal of software engineering. Reusing something that's already been built, tested, and paid for is one of the quickest ways to achieve this objective. From subroutines, to external libraries, to object orientation, to templates, to Web Services, each great advance in programming has been born from the desire to reuse material instead of starting from scratch. The limitation inherent in each of these milestones is that they were created by developers for the sole use by others in their profession.

It seemed inevitable that with the vast amount of new material being placed on the Web 2.0 every second, it could somehow evolve into raw material for software development. Tim Berners-Lee envisioned this leap in Web reusability in what he termed "the semantic Web," which describes a platform for the universal

exchange of data, knowledge, and meaning.[11] And while work continues to define new languages and protocols to realize Sir Tim's dream, mashups are making this vision a reality now.

Mashups are an empowering technology. In the past, resources had to be designed for reuse. Application program interfaces (APIs) had to be created, packages compiled, documentation written. The application developers and solution architects who recycled resources were subject to the whims of the original designers. With mashups, you aren't limited to reusing an existing API; you can *impose* your own if none exists. So if an application or site offers no API, or if you don't like the access methods that are already in place, you can design and implement your own (see the API Enabler pattern in Chapter 4 for several examples). The promise of achieving programmatic access to almost unlimited data is intoxicating. Even more exciting is the notion that the tools for constructing mashups have begun to reach a level of usability where even nontechnical users can build their own solutions.

Many popular definitions of a mashup would have you believe the term is limited to a combination of Web-based artifacts: published APIs, RSS/Atom feeds (see the "RSS and Atom" sidebar), and HTML "screen scraping." Although there are certainly valuable solutions in that space, a broader world of data can be mashed up, including databases, binary formats (such as Excel and PDF), XML, delimited text files, and more. The rush of vendors attempting to capitalize on the burgeoning market for enterprise solutions hasn't helped bring clarity to the field. To turn a classic phrase on its head, we have a ton of nails out there, and everyone is trying to tell us that they have the best hammer.

### RSS and Atom

RSS (also known as Rich Site Syndication or Real Simple Syndication) and Atom are formats for publishing Web-based content in a manner consumable by special applications termed "feed readers." Feed readers aggregate multiple feeds (or "subscriptions") so that a user can view updates to numerous Web pages from a single environment.

Before RSS and ATOM existed, users had to manually visit each site and check for any new updates. Feeds also serve as a popular method for allowing Web sites to dynamically incorporate content from external information providers. Regardless of their originally intended purpose, because feeds are created using a well-structured format (XML), mashups can easily consume them as a data source.

11. Berners-Lee, Tim, James Hendler, and Ora Lassila. "The Semantic Web." *Scientific American,* May 17, 2001.

Another common misconception is that mashups combine at least two disparate sites to form a brand-new "composite" application, complete with a neat new user interface. That's certainly possible, but mashups need not be an end in themselves. It is more accurate to say that all composite applications are mashups, but not all mashups are composite applications. The enterprise mashup creator can use the technology to transform the Web into his or her own private information source. This data can be used for strategic planning or analysis in systems like Excel or MATLAB. Mashups may also be used to access a single resource at superhuman levels to mine data or migrate content. Creating mashups is all about finding data, functionality, and services and using them to both solve problems and create opportunities.[12]

## Types of Mashups

Mashups have several different colloquial interpretations, which has resulted in some confusion regarding the term and its use. The word originated in the music industry, where a mashup was a combination of two or more songs to create a new experience. Typically, the vocal track of one song was combined with the instrumental background of another in this process.

The technology industry extended this definition to encompass a new application genus that described the combination of two or more sources into an integrated site. This technique of development hybridization can be roughly split into two separate categories: consumer mashups and enterprise mashups.

Consumer mashups are generally associated with Web 2.0. They require a lesser amount of programming expertise because they rely on public Web sites that expose well-defined APIs and feeds (see Figure 1.4).

The output is usually created by one of the sites participating in the mashup. In the classic "show craigslist listings on a Google map,"[13] the API of Google Maps is used to plot and present the feed obtained from craigslist.com. The limitation of this approach was that resources had to be "mashup ready."

Enterprise 2.0 mashups (sometimes referred to as data mashups) are more complex. Depending on which solution a firm deploys, enterprise mashups can emerge in several ways:

- Mashups are used solely by IT to rapidly deliver products. Application developers use both internal and external sources to create data mashups

12. This naturally presents potential legal complications, as discussed in Chapter 10.

13. http://housingmaps.com

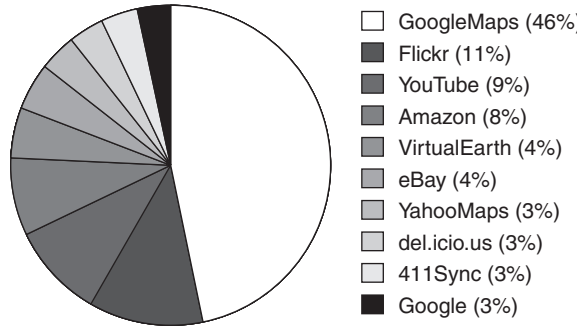



**Figure 1.4** *A small number of sites with public APIs account for the majority of consumer-created mashups. Source: http://www.programmableweb.com/apis*

and employ traditional coding techniques to create the user interface around them. Users aren't directly involved in the construction process but they benefit from IT's ability to provide solutions more quickly.

- IT creates a set of "mashable" components and gives end users a sand-box environment where they can freely mix and match the pieces together themselves. If users need new components, they have to solicit IT help to create them.
- An organization deploys an environment that lets anyone create and combine his or her own mashups. This approach is the most difficult implementation to manage, but probably has the greatest impact. To understand the challenge of this approach, consider the use of Microsoft Excel in many firms. Users can create spreadsheet-based applications and pass them around without any central oversight of what exists, how it is used, or if it was tested. This friction-free creation and distribution model spreads good solutions as quickly as bad ones.

Whether mashups are used by IT, business associates, or both, their agile nature makes them a key enabler of Enterprise 2.0. Unfortunately, they are not without potential downsides. In an attempt to "deconstruct" the success of Google, the *Harvard Business Review* points out several pitfalls[14] that can hinder success in a culture of open development:

- As people spend more time experimenting, productivity in other areas can suffer.

14. Iyer, Bala, and Thomas H. Davenport. "Reverse Engineering Google's Innovation Machine." *Harvard Business Review,* April 2008.

- Poor coordination across groups can lead to duplication of efforts and repeated mistakes.
- A constant stream of new products may confuse the organization and its employees.

Despite these potential hazards, the authors indirectly identify the virtuous circle of Enterprise 2.0 (Figure 1.5). As diverse products are combined to create useful new resources, they themselves become fodder for the next generation of useful products. In principle, this process isn't very different from the long-standing goal of reusability that firms have strived for in their applications and architecture. Three important differences arise this time around, however:

1. In the age of mashups "reuse" is no longer an ivory-tower concept restricted to the purview of application architects. Because end users and developers alike will be creating solutions, *everyone* will engage in the practice of reuse.

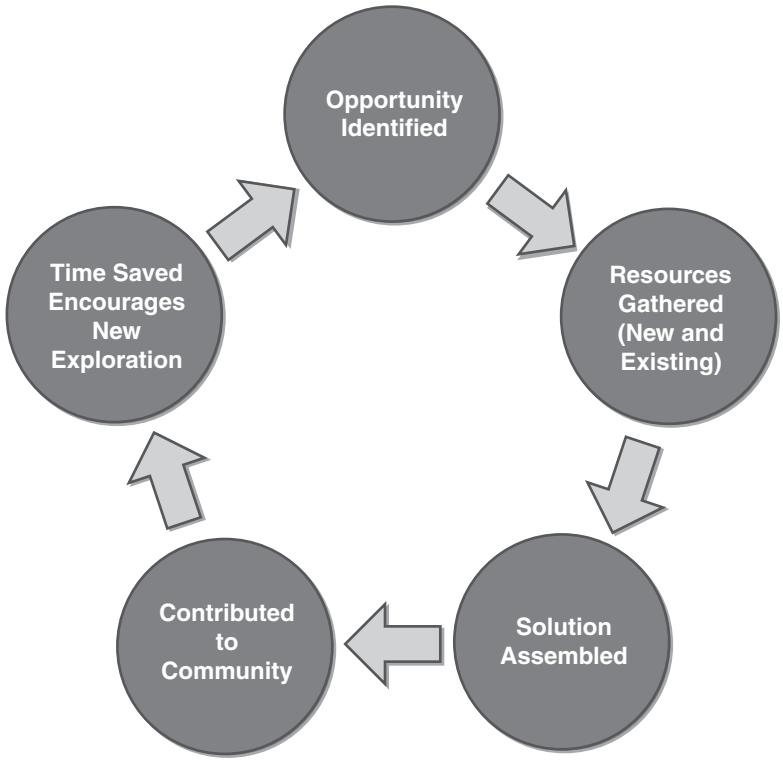



**Figure 1.5** *The virtuous circle of mashups*

2. The existing approach to reuse front-loads development efforts with additional planning and coding to create open APIs and extra documentation that may never be used. Because mashups impose reusability "after the fact," their creators will build their own APIs and include only the minimum functionality needed.

3. Traditional reuse practices don't require that a system that leverages existing code or libraries is itself reusable. This leads to implementations that are essentially "dead ends." Mashups are implicitly reusable, which creates a never-ending cycle of potential associations and recombination.

## Acquiring Data from the Web

Need input, More Input, MORE INPUT!
—Johnny Five, *Short Circuit,* 1986

As we saw in the last section, the majority of consumer mashups use the public APIs of a handful of Web sites. In the enterprise model, the best potential sources for mashup data may not be as forthcoming. In these situations, it becomes necessary to employ creative techniques to extract information. One of the most common and controversial techniques is often referred to as "screen scraping." This derogatory phrase carries a long sullied history and is thrown around by detractors seeking to undermine this approach.

Traditional "screen scraping" owes its origins to the early days of desktop computing, when IT departments developed various techniques to migrate "dumb terminal" mainframe applications to end-user computers. Rather than tackle the costly and time-consuming task of rewriting or replacing existing applications, many IT departments used special PC-based applications that emulated the original terminals.[15] These applications could receive the data from the mainframe and extract the contents of the forms presented on the old green-screen systems. User keystrokes were likewise emulated to send input back to the original application. This technique relied on developer-created templates and was both highly position-sensitive and extremely unforgiving. The smallest alteration in the mainframe display would break the predefined template and break the new application.

Because of these drawbacks, screen scraping was generally viewed as a hack and a last resort. The negative experiences associated with this approach continue to haunt any solution that promises to extract raw data from a user inter-

15. Such as an IBM 3270 or VT220.

face. Before organizations feel comfortable with mashups, users will need to understand how modern methods differ from the brittle approaches of the past.

Too many of us have forgotten that the "L" in HTML stands for "Language." In HTML, the *description* of the presentation and the *presentation itself* are inexorably bound in most people's minds. Many view HTML and what is displayed in their browser as two sides of the same coin.

In fact, it is the underlying Document Object Model (DOM) that makes mashup "screen scraping" something that should more appropriately be referred to as "Web harvesting" or "DOM parsing." When HTML is read by a browser, it is internally organized into a hierarchal structure. The underlying data structure is tree based and much more organized than what the user sees (see "The Structure of HTML" sidebar). HTML elements may contain additional nonvisual information such as the `id` and `class` attributes (see "The `class` and `id` Attributes" sidebar).

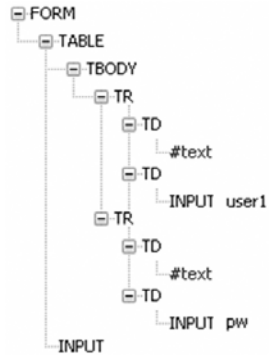
### The Structure of HTML

Consider the following simple Web form:


This is the underlying HTML:

```
<form method="POST">
<table border="0" width="250">
     <tr>
          <td width="85">User Name</td>
          <td><input id="user1" type="text" name="user_field" size="20"></td>
     </tr>
     <tr>
          <td width="85">Password</td>
         <td><input id="pw" type="password" name="password_field" size="20"></td>
     </tr>
</table>
<input type="submit" value="Logon" name="B1">
</form>
```

When parsed by a browser, this HTML is internally organized into a hierarchical structure known as the Document Object Model (DOM). The DOM is more conducive to automated analysis than the presentation users receive.




## The `class` and `id` attributes

The ubiquitous use of `id` and `class` in HTML make them ideal markers for Web scrapers to identify document elements.

**Uses of `id`:**

A style sheet selector

```
<P id="bigheader">Important Update</P>
```

A target anchor for hypertext links:

```
<H1 id="news">Today's Top Stories</H1>
```

A means to identify an element in JavaScript:

```
document.getElementById("news");
```

Used to name a declared OBJECT element:

```
<OBJECT declare
        id="newyork.declaration"
        data="city.mpeg"
        type="application/mpeg">
        A tour of Manhattan.
</OBJECT>
```

**Uses of `class`:**

Assign one or more CSS styles to an element:

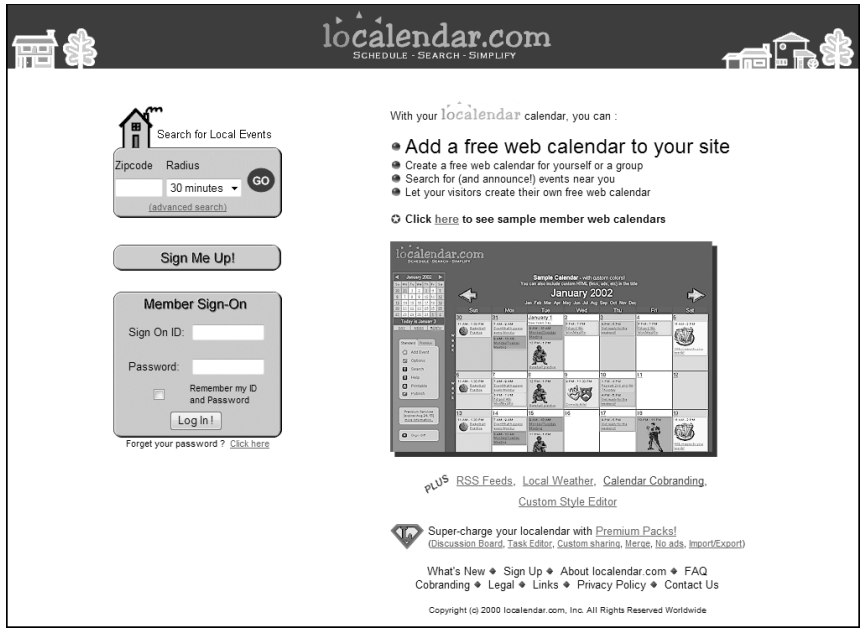
```
p.error {font-size: 18px; color: red;}
<p class="error">Incorrect Password</p>
```

Beyond their original intent within HTML, `id` and `class` attributes can also serve as "markers" for general-purpose processing by other applications/agents (e.g., mashups). Unlike the screen scrapers of the past that relied solely on positional information to parse screen content, mashups are able to examine the underlying attributes used to build the presentation. Although not a foolproof approach, this data changes much less frequently than the look and feel of a site, as demonstrated in the sidebar "Presentation Changes Don't Break Object Discovery." While consumer mashup builders queue up and wait for content providers to expose an API, enterprise teams are using Web harvesting to grab whatever data they want.

## Presentation Changes Don't Break Object Discovery

This example shows a sample Web page before and after a radical redesign. Although a visitor might be disoriented by the drastic changes, similarities in the underlying HTML (and resulting DOM tree) will not slow down a mashup that examines the site.

### Before




As part of a larger system, a mashup is created to sign in to a Web site by supplying a "Sign On ID" and a "Password." The form attributes and DOM information are displayed following the screenshot.

```
...
<td width="74" height="25"><div class="fixedfont">Sign On ID: </div></td>
<td width="89" height="25">
<p align="right"><input maxlength="20" name="username" size="10"
  style="font-family: courier"></p></td></tr>
<p align="center">
<tr>
<td width="74" height="5"><div class="fixedfont"> </div></td>
<td width="89" height="5"><div class="fixedfont"> </div></td></tr>
<tr>
<td width="74" height="25"><div class="fixedfont">Password:</div></td></p>
<td width="89" height="25">
<p align="right"><input maxlength="20" name="password" size="10"
  style="font-family: courier" type="password"></p></td></tr>
<tr>
...
```

**After**

Even though the site has been radically redesigned, it still contains form elements for "Sign On ID" and "Password." A peek at the underlying HTML and DOM shows that these fields retain the same attributes. A mashup most likely will not have a problem recognizing the new design, even though a human might take some time to become accustomed to the new interface.

```
...
<tr>
<td width="70" class="text_boxsubtitle">Sign-On ID:</td>
<td><input type="text" maxLength="20" name="username" size="10"
style='width:122px;FONT-FAMILY: Courier'/></td>
</tr>
<tr>
<td width="70" class="text_boxsubtitle">Password:</td>
<td><input maxLength="20" name="password" size="10" type="password"
style="width:122px;FONT-FAMILY: Courier"/></td>
</tr>
...
```

Enterprise mashups are not restricted to skimming content from HTML: They can leverage more structured formats such as XML (RSS, ATOM), Web Services, or even binary formats such as Excel and PDF (as shown in Figure 1.6). Nevertheless, the great promise of enterprise mashups derives from their ability to treat the entire World Wide Web as a first-class data source.

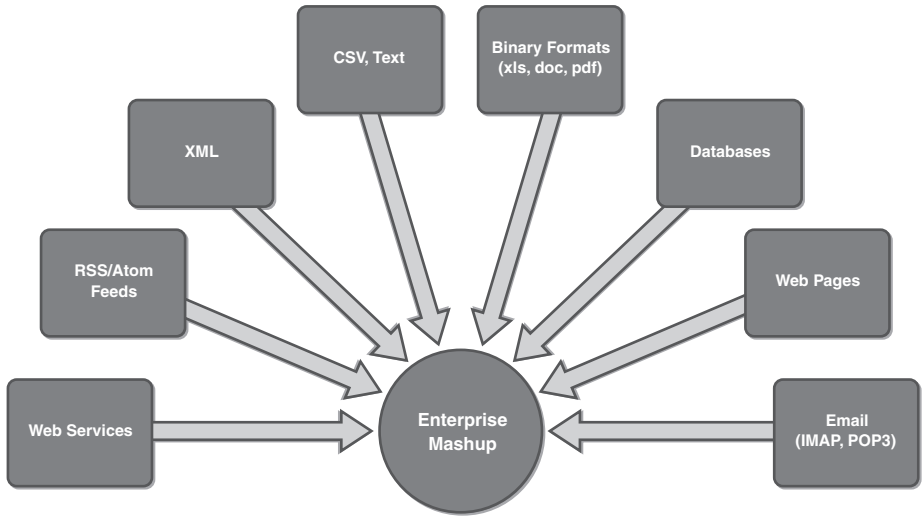



**Figure 1.6** *Enterprise mashups can consume a variety of different data sources.*

## The Long Tail

Although first coined to describe customers who purchase hard-to-find items,[16] the phrase "the Long Tail" has come to have a special meaning in the world of software. Traditionally, application development dollars are directed toward those projects and enhancements demanded by the largest group of users. This practice of catering to the masses doesn't necessarily lead to an outcome with the greatest positive impact on productivity. Unfortunately, because of the huge effort involved in developing applications, it is often impractical to provide custom solutions to a lone employee or a small team, even if it would greatly increase their efficiency (Figure 1.7). Thus only the "head" of the application demand curve is ever addressed. The exact cutoff point isn't fixed and will vary by organization, although the Pareto principle[17] or "80-20" rule suggests that 80% of application development efforts will benefit only 20% of your users.


IT focuses on the 20% of known problems that affect the most users (A).
The 80% of potential solutions (B) that serve a smaller audience are unaddressed.

**Figure 1.7** *The Long Tail*

The cumulative potential of unfulfilled Long Tail opportunities exceeds that of the "head" of the curve. Alas, fulfilling the requirements of the remaining 80% of your staff might seem an impossible goal. Most technology departments do not have enough staff to meet the needs of each individual user. Unless there is a way for developers to become drastically more productive or for end users to solve their own problems, the prospects for meeting unmet demand seem bleak.

16. Anderson, Chris. "The Long Tail." *Wired,* October 2004.

17. The Pareto principle is based on empirical observation and isn't a mathematical certainty in all cases.

## Meeting User Demand

Give me a place to stand on, and I will move the Earth.
—Archimedes

Enter the mashup. Armed with powerful new tools that leverage the resources of the Internet, developers and power users can quickly assemble products to target the Long Tail. We are witness to the dawn of a new era in technology. Mashups are making IT more agile and empowering individuals to create their own solutions.

The Long Tail is useful from an analysis standpoint only if it represents the universe of possible solutions that can be constructed. Consider the mashup example in "A Sample Mashup Use Case."

### A Sample Mashup Use Case

There are countless examples where mashups can benefit an enterprise, and they needn't be complex. Consider the following example.

Every day, the employees of a firm have numerous conference calls to discuss project planning, resource management, and corporate strategy. Whenever someone new joins the conference, there is a "beep" that announces that individual's presence. The first ten minutes of every call go something like this:

"Beep."
"Hi, who's on the line?"
"It's me, Rob."
"Beep."
"Hi, who's on the line?"
"It's me, Maureen."

On each call, valuable time is wasted while the moderator takes attendance and furiously scribbles down names. Later on, he may try and match those (frequently misspelled) names to an email address or telephone number.

We can save time and expedite the meeting with a simple mashup. First, we visit the conference call Web site and grab the participant's caller ID directly from the Web page. Next, we look up those numbers in the firm's online corporate directory (where we also get the corresponding email addresses). Finally, in case someone is dialing in from his or her home telephone, we use the search form on a public Internet site (such as whitepages.com) to look up any unresolved numbers.

The entire process is hidden behind a simple Web front end with a single button, labeled "Get Attendees." No more misspelled names or missed participants. No more pausing to ask latecomers to introduce themselves. Meetings start on time and everyone is happy.

As if this capability wasn't enough of a breakthrough, it opens up new possibilities for behavior tracking (also known as reality mining). You can click the "Get Attendees" button

multiple times during the call to see not only who is present, but *for how long.* Perhaps you can tie that "duration" data to other sources. You might find that callers drop off the line in coordination with weather, traffic patterns, or surf reports.

Although the "conference call attendance" issue was experienced by almost all employees of the firm, it was never identified as a business problem. This is because developers and business users are conditioned to view their actions in discrete, isolated chunks:

- First, I sign into Application A to locate a customer's account.
- Second, I sign into Application B to check item inventory.
- Third, I sign into Application C to create a purchase order for the client.

If you accept that Applications A, B, and C are immutable (perhaps because they were purchased from an external vendor), then you will never envision a solution where you can sign into Application D *once* and perform these three actions in a single step. The opportunity never appears on the Long Tail.

The greatest benefit of mashups may be their influence on our thought process. When we cast off our biases about the role of technology in the workplace, we discover the folly in applying IT to only the most obvious and well-understood problems. Once the blinders have been removed, you'll discover a world of missed and previously unknown challenges that you can tackle. Recognizing these opportunities is just the first stage. If you don't do something about them, then you've simply added to the tangle of unmet expectations. To achieve continuous innovation, it is essential to look outside the existing methods of measuring and meeting user demand.

## Mashups and the Corporate Portal

The concept of aggregating data from multiple sites inside and outside the workplace isn't new. As companies struggled to share all of their disparate applications and information resources directly with their employees, many embarked upon a quest to create a single corporate portal. An organization's portal typically provides several features:

- Single sign-on (SSO), which allows users to authenticate only once to obtain access to multiple applications.

- Multiple "portlets" or "islands" that expose information and functionality from disparate systems.
- Interaction (or integration), which allows portals to influence one another's behavior. For example, a search portlet may cause the contents of other portlets to be filtered.
- Access control, which provides for the centralized administration of which information a user may access. A user's permissions on the portal are at least as restrictive as what the user would receive if he or she logged into the underlying application directly. Portals are unique in that they may bring content together from multiple sources wherein the user has varied entitlements.
- Personalization, which allows the user limited ability to customize the layout and presentation of the site to suit his or her own specific tastes and needs.

Of course, as our examination of the "80-20" rule suggests, portals will never meet the requirements of all users, all of the time. At best, they may meet the lowest set of common requirements across a broad audience (the 80%). The most specific requirements are typically the least general (the 20%), which explains why most corporate portals typically confine themselves to broadcasting company news, managing health and benefits information, and tracking the holiday calendar. Personalization, the latecomer to the portal infrastructure, was a desperate attempt to address this shortcoming. Unfortunately, users typically don't get a say in choosing *which* content can be personalized or *how* it can be manipulated.

At my daughters' nursery school, their teacher maintains order by telling the children, "You get what you get and you don't get upset." Those days in computing are passé. Whether we are talking about the corporate business user who wants to come to the office each day to a personalized workstation or a customer who wants to view your company's information in a certain fashion that suits his Web-based applications, this is the age of individualized construction.

When the popular social networking sites MySpace and Facebook published open APIs to leverage their data and create interfaces around it, thousands of users became bona fide developers. They quickly learned to build their own personal portals. This same demographic is just now beginning to enter the Enterprise 2.0 workforce. They won't be content to operate within the confines of a single, stoic portal that restricts how they consume and manipulate information.

A new metaphor for user interaction has recently emerged that, combined with mashups, threatens the relevance of the enterprise portal. Whether you know them as widgets, gadgets, or snippets, they are the small plug-in components that originated on the Web and have migrated to the desktop (e.g., Apple Dashboard, Yahoo Widgets, Google Gadgets, Microsoft Vista Desktop Widgets).

The tools for creating these "mini-applications" have become easier to use and more familiar to a much broader audience.

If enterprise mashups are the path to user-created data and widget platforms are the environment for presenting that information, the combination of the two represent the death knell for the corporate portal. At best, it will morph into a set of core services that provide information to mashup-powered personal environments.

## Mashups and Service-Oriented Architecture

Service-oriented architecture (SOA) has come to be associated with Web Services, but at its core it is more mindset than methodology. The "service" in SOA shouldn't be thought of in terms of a particular technology, but rather as a *business task*. The tasks are implemented in an environment that facilitates loose coupling with other services. This combination, in turn, fosters an atmosphere where developers can create new applications that reuse and recombine existing functionality. Because the services are based on open standards, they can be consumed equally well across independent development platforms.

The promise of SOA is that it addresses the Sisyphean[18] labor of building duplicate, siloed functionality across the enterprise. Better yet, you don't have to build services yourself; you can discover and use third-party solutions. SOA is the equivalent of a home improvement store for application development. You simply fill up your shopping cart with all the raw materials and glue and nail them together in your basement to create a shiny new product. Using a traditional development mindset would place the burden on you to chop down trees for lumber or smelt the iron for nails.

The Common Object Request Broker Architecture (CORBA) was an early stab at implementing SOA—so early, in fact, that it predates the Internet explosion of the mid-1990s and even the SOA acronym itself. The level of complexity required to work with this technology was often found to outweigh its benefits, and while CORBA struggled to find its footing, newer technologies such as SOAP, XML, and Java (Enterprise Java Beans) arrived on the scene. They began to address the problems associated with CORBA's steep learning curve and security shortcomings.

18. Sisyphus was a Greek who was condemned by the gods to ceaselessly roll a rock to the top of a mountain, only to have it fall back of its own weight.

Web Services emerged as a technology-agnostic interoperable solution based on open standards such as XML, WSDL, UDDI, and SOAP. Although far from perfect,[19] SOAP-based Web Services have become the industry-preferred method for implementing SOA. The most popular method for exposing SOAP services across the enterprise is via a custom infrastructure known as an enterprise service bus (ESB). The ESB can provide additional data transformation capabilities, security, transaction support, and scalability, all while simultaneously reducing the degree of complexity exposed to service reusers. In an attempt at product differentiation, some ESB offerings service-enabled existing corporate resources (such as databases) and were themselves progenitors of the data mashup.

One point should be clear: SOA is not a revolutionary milestone but an evolutionary one. Open communication and technology standards, combined with the ubiquity of the protocols that power the Web, have finally helped SOA reach a level of maturity where its benefits exceed its costs.

Mashups represent the next leap in reuse. They initially came about when developers combined the published APIs of different Web applications to create interesting new content. The limitation of this approach was that resources had to be "mashup ready." Robust SOA environments were a hothouse for mashup growth, as they exposed componentized functionality that could be mixed together to provide new services.

You may be wondering if mashups are the latest harbinger of SOA, or the beneficiary of it. The answer is a resounding "Both!" With most vendors now using the terms "SOA" and "Web Services" interchangeably, it has become obvious that for most corporations, implementing a successful SOA will require the service-enablement of their existing applications. Mashups are a completely valid method of accomplishing this (see the "API Enabler" section in Chapter 4 and the discussion of the Quick Proof-of-Concept pattern in Chapter 7). Most mashup products allow you to create and publish Web Services either directly or via a third-party application container (e.g., WebSphere or JBoss). Likewise, mashups are voracious consumers of Web Services. Mashups gladly leverage the Web Services that SOA-centric organizations already have in place. Because mashups can produce services with the same agility that they consume them, they are a valuable addition to any service-oriented environment.

How do SOA patterns and mashup patterns relate to each other? SOA generally focuses on server-side architecture and internal corporate resources, whereas everything is fair game with mashups. Because of SOA's maturity and

19. Problems include interoperability issues and platform-specific implementation, testing, and security challenges.

association with Web Services, it has achieved greater clarity regarding its capabilities, protocols, implementation, and use. This allows SOA pattern discussions to focus on high-level abstractions. Indeed, several excellent Web sites and books[20] discuss the process of SOA-enabling the enterprise. Mashup patterns, which remain in a nascent stage of development, must focus on more practical examples. This will drive broader adoption, which in turn should to lead to consolidation and standardization similar to what SOA has achieved.

## Mashups and EAI/EII

Enterprise application integration (EAI) is the practice of connecting corporate systems at the application level rather than at the data level. EAI solutions seek to streamline business processes and transactions, whereas mashups typically combine applications with the goal of providing new functionality. EAI tools rely on support for open standards such as Web Services or CORBA. If an application doesn't expose an API, one needs to be constructed programmatically. As systems and requirements evolve, there is an inevitably large carrying cost to maintain the custom integration code. When managed and funded correctly, EAI can provide the most rock-solid method of application integration. For business-critical solutions, EAI is recommended over mashups, which permit some fragility as a trade-off for the benefit of agility.

Enterprise information integration (EII) is a data management strategy for providing uniform access to all the data within an organization. The rise of "big box" stores that sell everything from baby clothing to car tires has demonstrated that patrons appreciate the convenience of one-stop shopping. Collecting data from multiple sources and providing a single point of access has similar appeal in the enterprise. EII is often easier to achieve than EAI because it simply attempts to unify information and not applications. If you think this approach sounds similar to a data mashup, you're correct. A mature EII implementation can provide new insights into data associations and facilitate rapid solution delivery. EII tools have historically focused only on back-end databases,[21] which limits the range of information that can be collected. By comparison, mashups surpass EII in their ability to obtain data from both structured and unstructured sources.

20. Author Thomas Erl has written several good books on this subject, including *SOA Design Patterns*.

21. These databases include relational databases, message queues, and data warehouses.

The knowledge requirement for successfully applying EII technology is higher than that for mashups, but as with EAI the advantage is stability. You can measure the benefits of a complex EAI/EII project empirically by developing a quick mashup-based prototype (see "Quick Proof-of-Concept," Chapter 7). This effort may help determine whether the potential benefits justify the considerable cost and time required to carry out a formal implementation.

## Mashups and Software as a Service

In contrast to the architectural style and Web Service implementation strategy of SOA, software as a service (SaaS) is a business model. SaaS is the latest incarnation of the Internet-boom idea of an application service provider (ASP). Under the SaaS plan, businesses do not invest money to develop and host applications internally, but instead rent the functionality they need from an external service provider. End-user interaction with applications typically occurs via a prebuilt Web interface. The customer's business data is then fed into the system manually, using Web forms, or programmatically, using a Web Service API.

To appeal to as broad a market base as possible, most SaaS providers have focused on generic services and priced them competitively (a fee of less than $100 per service is not uncommon). Exposing macro capabilities and parameterizing functionality allows customers to achieve some degree of customization.

One of the most prominent success stories in SaaS is Salesforce.com. This "zero-infrastructure" customer relationship management (CRM) platform provides services to thousands of businesses worldwide. Small and large customers alike are able to start using the hosted service almost immediately without deploying custom hardware. The success of Salesforce.com has led many to assume SaaS is particularly well suited to CRM and sales force automation. In reality, this isn't the case. WebEx, a Web-based conference and collaboration solution, has achieved adoption on an even larger scale. Google Apps is an example of a viable alternative to traditional desktop software. It serves up a business-focused mail, spreadsheet, and word processing suite at a fraction of the cost of Microsoft Office. Many commercial vendors are exploring SaaS to create new revenue streams.

Assuming SaaS products can meet technical and functional user requirements, two key challenges must be overcome before SaaS can succeed as a general distribution model. First, firms must be comfortable with the notion that their data is housed externally to the organization. It seems that there's a new story almost every day in the press about missing hard drives or accidentally

leaked personal information. SaaS providers may have better security than many of their clients, but the abdication of data management to a third party is still a tough pill for many corporations to swallow. The second obstacle for SaaS is availability. For mission-critical applications, the network remains a potentially dangerous point of failure.[22]

Mashups are a natural complement to SaaS. Perhaps there are SaaS solutions that appeal to your organization, but you have held back on implementing them because you couldn't get exactly the functionality you required from a single provider. Maybe the SaaS product is extensible, but you don't want to invest time and money in duplicating functionality you've already built internally. Mashup patterns such as Workflow (see Chapter 5) and Content Integration (see Chapter 6) can be used to link an external solution and internal products together. With SaaS and mashups, you may be able to maintain the bulk of your confidential data internally and send the hosted application only small subsets of data for processing. If the network link to the SaaS vendor fails, at least you will still have local access to your data.

If you're thinking about testing the SaaS waters as a *vendor,* then applying SOA via mashups can help you get started. The API Enabler (see Chapter 4) and Quick Proof-of-Concept (see Chapter 7) patterns are excellent means of creating a Web interface to your existing resources. You can use the Load Testing pattern (see Chapter 8) to see how your systems scale under heavy user activity.

SaaS shares another characteristic with mashups: It may already be in use in your company without your knowledge. Because this model requires only a Web browser and no special infrastructure, it is easy for end users to circumvent IT and obtain applications directly. It is crucial that an IT department doesn't have a monitoring and enforcement policy based solely on policing internal data centers. IT personnel need to engage with the business users and educate them about the risks and rewards of SaaS and the effects these decisions will have on future growth. Internal checkpoints with purchasing and legal departments are a necessity, too. All service level agreements (SLAs) should be reviewed and signed by appropriate parties, and attempts to expense software purchases that have not been vetted by IT should raise a warning flag. Otherwise, SaaS can sneak into your organization on a corporate credit card.

22. Service level agreements (SLAs) should be in place to ensure your applications are available when needed.

## Mashups and the User

Make no mistake about it—despite the recent buzz around Enterprise 2.0, people have been creating mashups for many years. Of course, the process to this point has been overwhelmingly manual. Microsoft Excel is arguably the father of the corporate data mashup. For years, Excel end users have cut-and-pasted data to feed their calculation engines. Spreadsheet-based solutions have spread throughout the enterprise without the involvement of IT. Mashup tools enable the automation of this aggregation process, and a new clan of users is poised to run wild with the technology.

A culture of individualism is clearly emerging in today's world. People no longer plan their evenings around what TV networks schedule for them to watch, for example. Instead, they record their favorite shows onto digital video recorders (DVRs) or watch movies and shows on their computers and mobile devices. Similarly, the recording industry no longer has a stranglehold over music distribution. Newspaper readership is down, as more individuals choose to consult RSS feeds and blogs instead of purchasing the printed documents. People can even create personalized clothing and sneakers online.[23] Members of the public have evolved from docile consumers into "prosumers."[24] Products and services are moving away from mass markets and being shaped by the people who consume them. Likewise, a fundamental shift has occurred in software development. Armed with new tools and the skills to use them, users aren't waiting for IT to build solutions—they're doing it themselves.

Should organizations facilitate these individuals' efforts, or rein them in? For years, the mantra of professional software development was "Separate business logic from presentation logic." Programmers religiously structured their code around that principle but ignored the logical conclusion: *The best shepherd of business expertise is not the IT department, but the business users themselves.*

The inclination for IT departments to view user-led efforts in an adversarial light increases when IT experts believe that their "home turf"—application development—is threatened. IT needs the occasional reminder that in any development effort, it is the users who are the key to defining metrics for success. Besides, users are already creating mashups anyway, albeit human-powered ones.

23. Nike iD lets you design custom shoes and clothing (http://nikeid.nike.com).

24. Toffler, Alvin. *The Third Wave*. 1980.

Gartner has said mashups will make IT even more critical to business operations,[25] so a knee-jerk rejection to their emergence is not necessarily in the best interests of the firm. Rather than deny business users the tools that can increase their productivity, IT needs to embrace a new model. Besides, starting with a mashup product won't get you a business solution any more than staring at a word processor will get you the next great novel.[26] Because IT personnel clearly cannot scale to meet the requirements of each particular user, they should leverage the potential of mashups and work in partnership with the business associates to train a new class of self-serve builders. This effort is akin to adding hundreds of new developers at virtually no additional cost.

It's a common assumption that the latest generation of developers is intuitively suited to filling this role. Affectionately termed the "Millennials" or "Generation Y," these individuals came of age during the Internet boom of the last decade and are inherently comfortable with technology. Millennials, green with inexperience and giddy about tinkering, question everything. This behavior stands in stark contrast to that of the entrenched workforce, whose habits of working in a particular manner condition them to no longer question the "why."

Many companies are rushing to embrace Web 2.0 ideals such as mashups, social networks, wikis, and blogs not because they have inherent value, but rather because the firms think this practice will attract the "new thinkers." In reality, instead of abdicating responsibility for innovation to a younger generation or applying technology Band-Aids, firms need to cultivate an environment of creativity and collaboration for their employees regardless of their physical age. Any firm can realize the value of mashups and Enterprise 2.0 so long as its managers are capable of taking a critical look at their workplace and realizing they don't need to settle for "good enough" any more.

The "guerrilla-style" approach of mashup development is not without its drawbacks, of course. Most business users do not fully grasp the challenges in providing scalability, reliability, business continuity, disaster recovery, security, and fault tolerance. If users are permitted to develop ad hoc solutions, IT must provide an environment that cultivates these best practices.

## A Patterns Primer

The benefits of enterprise mashups are communicated through a concept known as a *pattern*. If you've ever baked holiday cookies, then you already

25. David Cearley, Gartner analyst.

26. Or a *Mashup Patterns* book—trust me, I've tried.

have some idea of what a pattern is and how it works. Suppose you want to make a tray of chocolate-chip heart-shaped cookies. After you mix the dough, you roll it out and grab your cookie cutter. You use the cutter to press out a series of identical shapes. Afterward, you decide some oatmeal raisin hearts would be nice, so you mix a fresh batch of ingredients and cut out another series of hearts. The cookie cutter is a form of pattern. The different types of dough are the specific situations, or "use cases," where the pattern is applied. A pattern doesn't solve a problem in itself. It's just a general form that helps you think about the structure of the solution (what shaped cookie, in this example).

The remaining chapters of this book present a number of patterns, along with some examples to illustrate how they work in an enterprise context. Don't throw out the pattern if you don't like the dough! Every business has a different flavor, and the key to success with patterns is figuring out which one is yours. You can use the samples that fill out this book to help identify the mashup ingredients your organization already has. Apply the appropriate mashup pattern and you have a recipe for success.[27]

## The Fragility Factor

It may seem that the title of this book is an oxymoron. How can something as ad hoc and unstructured as Web scraping be coupled with something so formal and structured as a pattern? Ideally, the previous discussion of how mashups work under the hood will have made you more comfortable with the technology.

If you think reverse-engineering Web pages still doesn't sound like the type of rock-solid approach that a professional developer should be using, I don't blame you. One of the core tenets of software engineering is that applications should behave in a reliable and predictable manner. Web harvesting—although a great deal more reliable than screen scraping—is inherently unstable if you don't control the Web sites from which you extract data. Because you can't determine when a scrape-based solution might break, you should never employ this approach on a mission-critical system.

If you have the chance to help your firm gain a competitive advantage or reduce costs—even if just for a limited time—you should explore the opportunity.

27. The classic reference for pattern-based design is Christopher Alexander's seminal text *The Timeless Way of Building* (Oxford Press, 1979). Buildings, like software components and cooking ingredients, can be combined in an almost endless variety. Nevertheless, certain basic concepts govern which elements work well together and which don't.

*There is nothing wrong* with an application that has a short lifespan, so long as you don't create a situation where the cost of remediating or retiring the solution exceeds the achieved benefit. The rapid speed with which mashups can be developed means occasional remediation isn't a time-consuming task. Plus, quick release cycles translate into more chances for exploratory development, which in turn can lead to the discovery of new uses or solutions.

The patterns in this book all adhere to this basic premise. You won't find examples of settling stock trades or sending online payments, even though mashups can facilitate those tasks. It's simply irresponsible to use the technology in this manner. Like any development effort, a mashup solution will require regular maintenance over its lifetime. Unlike with traditional applications, you may not be able to determine the time when this work will be required. Web Service APIs can change, RSS feeds can be restructured, or site redesigns may temporarily toss a monkey-wrench into your application's internal workings. Because of these possibilities, you should implement mashup-based solutions only where you can tolerate temporary downtime that may occur at unexpected intervals.

The fragility score is an ad hoc[28] rating based on a number of factors:

- A mashup pattern that relies on a single Web site (e.g., Infinite Monkeys, Time Series, Feed Factory, Accessibility, API Enabler, Filter, Field Medic) is less fragile because there is only a single point of potential failure.
- A multisite-based pattern (e.g., Workflow, Super Search, Location Mapping, Content Migration) is more fragile with each additional site that it leverages.
- Mashups that employ Web harvesting are generally more fragile than those that use feeds (RSS, Atom). Feeds are, in turn, more fragile than Web Service APIs. APIs are the most stable integration point because they reflect a site's commitment to expose data and functionality.
- Mashups that mine data from "hobby" sources have a greater risk of failing. For example, obtaining local weather data from the U.S. government-funded National Oceanic and Atmospheric Administration's (NOAA) weather site (http://www.nws.noaa.gov/) is probably a safer bet than obtaining the information from your local high school or radio station.

28. Translation: "Your mileage may vary." The fragility score is based on unpublished observations of the technology and will vary according to the resources you incorporate in your specific implementations.

For-profit sites may exert legal pressure to halt mashups (see the Sticky Fingers anti-pattern).

- Mashups that use boutique data not widely available on the Internet are at high risk. What are your alternatives if the site suddenly vanishes one day?

Each pattern template described in this book contains a fragility score ranging from 1 glass  (the least fragile) to 5 glasses  (the most fragile). No pattern receives a score of zero, because even the most rigorously tested mashup-backed application always has some degree of brittleness.

The fragility score is ultimately intended to encourage *thought* about mashup stability. It's possible to have five sites in a multisite pattern that change less frequently than an individual Web page used in a single-site pattern. This is particularly true when vendor products and internally created systems are involved. The user interfaces of commercial and in-house applications aren't frequently redesigned. Public Web sites, in contrast, must constantly reinvent themselves in the battle to attract eyeballs.

If you create a mashup-based solution and don't acknowledge that it encapsulates some degree of uncertainty, you are just kidding yourself. Worse, you are deceiving your users, who will not be pleased when the system "mysteriously" fails one day.

In case you think only mashups have this Achilles' heel, keep in mind that any distributed system (which is what a mashup is) contains an inherent level of risk. Each additional component and the infrastructure that connects it represent another potential point of failure. So before you think, "Why the heck would I build something that might break?" consider how you have handled similar situations in the past. You can address many of these fragility issues by thinking about redundancy, monitoring, and notification up front.

## The Future of Mashups

Mashups aren't just about mixing Web sites together to create new solutions—they're a tool for unlocking the treasure chest of data right under your nose.

The primary goal of this book is for the reader to scan at least one pattern and realize, "I never thought you could do that!" The examples that accompany the patterns are aimed at both the business end user and the technical user. When you understand how mashups can be used to mine new information or automate traditionally manual activities, you'll never look at your workplace in

quite the same way. The morass of daily problems suddenly becomes visible—but now you'll have the inspiration and knowledge to tackle them. As with the classic *Design Patterns* text, *Mashup Patterns* is intended to provide a general language that developers and the business can use to succinctly communicate about a solution ("Oh, we can use a Time Series mashup here").

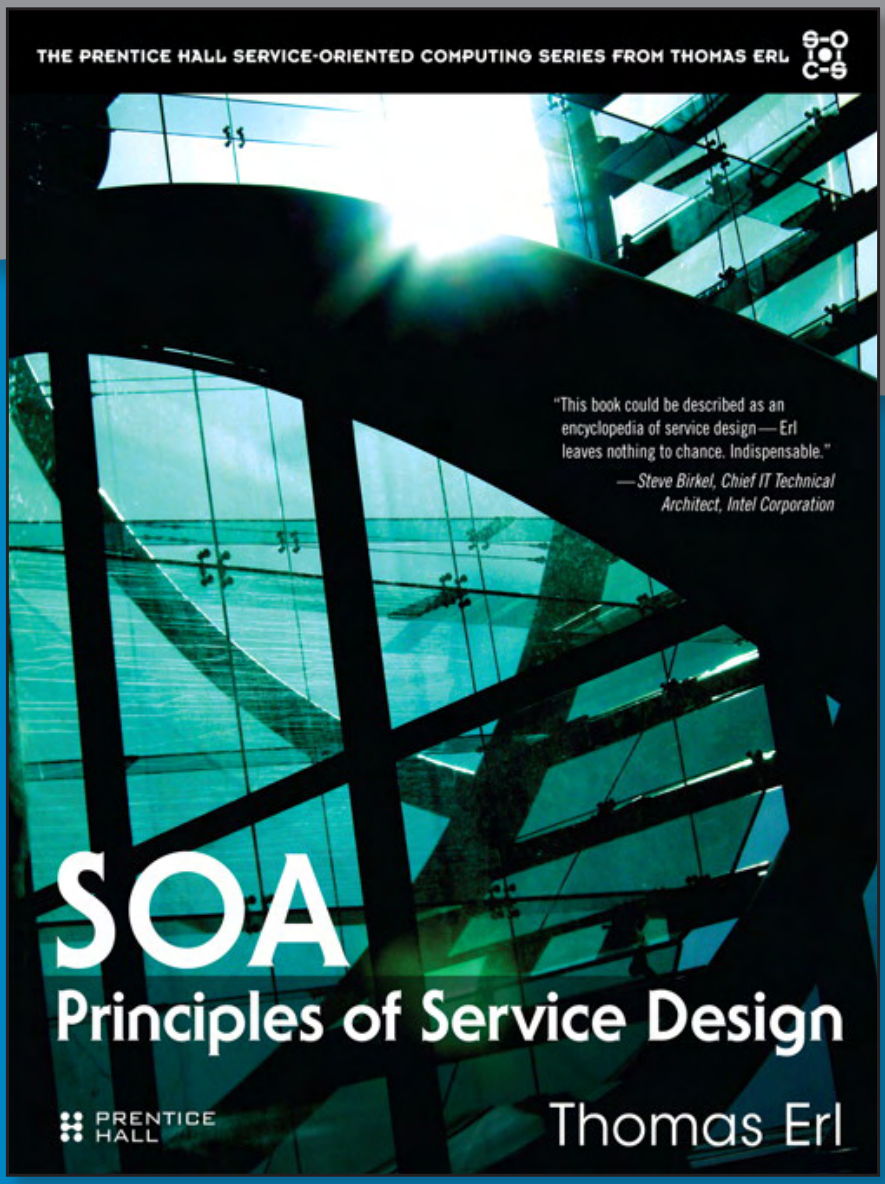
It's not every day that we witness a groundbreaking advancement in application development. Most improvements occur gradually and can take years to snowball into something useful. They may require costly new investments in infrastructure or reengineering of existing resources. Or they may be confined to a narrow niche in our industry. Only the naive overlook the dangers that come with any great leap; only the foolish cite those risks as reason enough to ignore the potential benefits.

Don't let the hype surrounding mashups cause you to abandon the best practices that guide good development. Likewise, be open to thinking creatively about the problems that exist around you. Employees who face seemingly intractable problems or whose careers have trained them to ignore the breakdowns in their organization will be delighted to discover that practical solutions are now available. The patterns in this book will help you get started by demonstrating how mashups can help you achieve the following goals:

- Make money for your organization
- Fill gaps not met by the existing IT infrastructure
- Create a quick proof-of-concept to explore new solutions
- Gain a competitive advantage
- Avoid "information overload"
- Expose your applications to a wider audience

and more!

Enterprise 2.0 is all about You. And the potential benefits from mashups are as big as anything you can imagine.



Thomas Erl

BUY ME


# SOA Principles of Service Design

## The Definitive Guide to Service Engineering

The key to succeeding with service-oriented architecture (SOA) is in comprehending the meaning and significance of its most fundamental building block: the service. It is through an understanding of service design that truly "service-oriented" solution logic can be created in support of achieving the strategic goals associated with SOA and service-oriented computing. Bestselling SOA author Thomas Erl guides you through a comprehensive, insightful, and visually rich exploration of the service-orientation design paradigm, revealing exactly how services should and should not be designed for real-world SOA.

THIS BOOK'S IN-DEPTH COVERAGE INCLUDES

- Over 240 full-color illustrations.
- A concise introduction to SOA and service-oriented computing concepts and benefits.
- A thorough exploration of the service-orientation design paradigm as represented by eight specific design principles.
- A comparison of service-oriented and object-oriented concepts and principles and a clear definition of what qualifies as "service-oriented" logic.
- Detailed coverage of four different forms of service-related design granularity.
- An exhaustive examination of service contracts, with an emphasis on standardization, abstraction, and the utilization of WS-Policy, XML Schema, and WSDL definitions.
- A comprehensive study of positive and negative service-related coupling types with an emphasis on the requirements to attaining a suitable level of loose coupling.
- An inside look into how commercial design approaches are incorporated to achieve truly agnostic and reusable service logic.
- Techniques for maximizing service reliability, scalability, and performance by instilling high levels of autonomy and emphasizing stateless design.
- Approaches for positioning services as highly discoverable and interpretable enterprise resources.
- Unprecedented coverage of how to design services for participation in complex compositions.
- The definition of concrete links between each design principle and the strategic goals and benefits of SOA and service-oriented computing.
- Numerous cross-references to key design patterns documented separately in *SOA: Design Patterns*.

AVAILABLE
- BOOK: 9780132344821
- SAFARI ONLINE
- EBOOK: 0132361132
- KINDLE: B001334IXI

## About the Author

THOMAS ERL is the world's top-selling SOA author, *Series Editor of the Prentice Hall Service-Oriented Computing Series* from Thomas Erl, and Editor of The *SOA Magazine* (www.soamag.com).With over 100,000 copies in print worldwide, his books have become international bestsellers and have been formally endorsed by senior members of major software organizations, such as IBM, Microsoft, Oracle, BEA, Sun, Intel, SAP, CISCO, and HP. Thomas is the founder of SOA Systems Inc. and the internationally recognized SOA Certified Professional program (www.soacp.com). Articles and interviews by Thomas have been published in numerous publications, including *The Wall Street Journal* and *CIO Magazine*.

PRENTICE HALL

informit.com/ph

# Chapter 5

# Understanding Design Principles

Principles help shape every aspect of our world. We navigate ourselves through various situations and environments, guided by principles we learned from our family, society, and from our own experiences. Historically, many parts of the IT world encouraged the use of design principles so that when you did something, you would "do it right" on a consistent basis. Often, though, their use was optional or just recommended. They were viewed more as guidelines than standards, providing advice that we could choose to follow.

When moving toward a service-oriented architecture, principles take on renewed importance primarily because the stakes are higher. Instead of concentrating on the delivery of individual application environments, we usually have a grand scheme in mind that involves a good part of the enterprise. A "do it right the first time" attitude has therefore never been more appropriate. SOA projects have the potential to shape and position solution logic in ways that can significantly transform an enterprise. We want to make sure we steer this transformation effort in the right direction.

As documented in Chapter 4, the design principles explored in this book establish a paradigm with many roots in previous computing generations. None of them are really that new. What is distinct about service-orientation is which of these existing principles have been included and excluded—that and the high-minded goals promised by its successful application.

## 5.1 Using Design Principles

The *Design Fundamentals* section of Chapter 3 formally defined the term "design principle" and determined that it essentially is "a recommended guideline for shaping solution logic with certain goals in mind." We subsequently covered the following list of service-oriented computing benefits:

- Increased Intrinsic Interoperability
- Increased Federation
- Increased Vendor Diversification Options
- Increased Business and Technology Domain Alignment

- Increased ROI
- Increased Organizational Agility
- Reduced IT Burden

These benefits represent the most common strategic goals associated with service-orientation. The application of the eight principles explored in this book results in the realization of very specific design characteristics, all of which support these goals.

We therefore need to ensure that the principles are effectively applied. Following is a set of best practices for getting the most out of the design principles in this book.

#### Incorporate Principles within Service-Oriented Analysis

Because we have labeled the principles in this book as *design* principles, there is a natural tendency to focus on their application during the design stage only. However, because of the unique form of analysis carried out as part of the common SOA delivery lifecycle, it can be highly beneficial to begin working with a subset of the principles during the analysis phase.

While iterating through the service modeling process of a typical service-oriented analysis, we are tasked with defining a conceptual blueprint for the inventory of services we will eventually be designing and building. This provides us with an opportunity to begin conceptually forming some of the key service design characteristics ahead of time.

Of the eight service-orientation design principles, the following three are most commonly incorporated within the service modeling process:

- *Service Reusability*—Reusability considerations are highly relevant to defining the inventory blueprint because they help us group logic within the contexts of proposed agnostic service candidates and further encourage us to refine the definition and functionality behind agnostic capability candidates.
- *Service Autonomy*—One of the goals of the information gathering steps that comprise the parent service-oriented analysis process is to determine where, within an enterprise, autonomy will ultimately be impacted. Knowing this in advance allows us to adjust service candidate granularity and capability candidate grouping in response to practical concerns. This prevents the inventory blueprint from becoming too abstract and out of touch with the realities of its eventual implementation.

- *Service Discoverability*—Although service meta data can be added to a contract at any time prior to deployment, the analysis stage enables us to leverage the expertise of subject matter experts that will not be participating in subsequent project phases. This is particularly relevant to the definition of business services. Analysts with a deep insight into the history, purpose, and potential utilization of business logic can provide quality descriptions that go far beyond the definition of the candidate service contract.

As illustrated in Appendix B, a separate step dedicated to applying select service-orientation principles is part of a standard service modeling process.

**FOR EXAMPLE**

A US-based shipping company created their own expanded variation of the service modeling process documented in Appendix B. Instead of bundling service-orientation considerations into one step, it included the following separate steps:

- *Business Reusability Survey*—A step during which representatives from different business domains were questioned as to the applicability of a given service that was being modeled. Those surveyed were asked to provide feedback about how any agnostic service could be potentially extended in support of business processes that resided in their domains.
- *COTS Evaluation*—This was carried out for each service capability candidate required to encapsulate functionality that resided in an existing COTS environment. It provided insight into potential autonomy constraints for some of the planned services.
- *Service Profile Copyedit*—This was a step toward the end of the modeling process during which the service profile document was refined by one of the on-staff technical writers (which is also a best practice discussed in Chapter 12).

Each of these steps was carried out by different individuals, all part of the service modeling project team.

### Incorporate Principles within Formal Design Processes

The key success factor to leveraging service-orientation design principles is in ensuring that they are applied consistently. When services are delivered as part of different projects that are perhaps even carried out in different geographical locations, there is a constant danger that the resulting service inventories will be comprised of incompatible and misaligned services, varying in both quality and completeness.

Design synchronicity is important to achieving the harmonization and predictability required to ultimately compose services into different configurations. Establishing formal service design processes that exist as part of the organization's over-arching project delivery methodology requires that project teams give serious thought as to how each principle can or should be applied to their planned services.

The design processes listed in Appendix B have steps dedicated to applying service-orientation principles. These processes can be further customized and expanded to incorporate a dedicated step for each principle.

**FOR EXAMPLE**

The aforementioned shipping company formalized service design processes that included separate steps for applying Service Reusability, Service Autonomy, and Service Composability principles. The remaining principles were also incorporated in the design processes but grouped together with other design considerations.

The Service Composability step actually introduced a sub-process during which service contracts were combined into a variety of composition configurations in order to assess data exchange compatibility.

### Establish Supporting Design Standards

Design principles are design guidelines, essentially recommended approaches to designing software programs. Due to the importance of creating consistent programs (services) in support of service-oriented computing, it is highly recommended that design principles take on a larger, more prominent role.

Once an organization has determined to what extent it wants to realize service-orientation, design standards need to be put in place in full support of the consistent application of these design principles. This often leads to the principles themselves forming the basis for multiple design standards.

Either way, if you are expecting to attain meaningful strategic benefit from a transition toward SOA, design standards need to be in place to ensure the consistent realization and proliferation of service-orientation across all affected services.

**FOR EXAMPLE**

An enterprise design specification for a government agency contained upwards of 300 separate design standards, many of which were directly or indirectly defined in support of service-orientation.

One of these standards, for example, required that all XML schema definitions support null values by allowing an element to exist zero or more times (via the `minOccurs="0"` attribute setting). If the element was not present, its value was considered to be null.

This simple design standard ensured that null values were consistently expressed across all XML document instances, thereby supporting the Service Contract Standardization and Service Reusability principles and also avoiding some of the negative coupling types described in Chapter 7.

### Apply Principles to a Feasible Extent

Each of the eight service-orientation design principles can be applied to a certain extent. It is rare that any one principle will be fully and purely realized to its maximum potential. A fundamental goal when applying any principle is to implement desired, corresponding design characteristics consistently within each service to whatever measure is realistically attainable.

The fact that principles are always implemented to some extent is something we need to constantly keep in the back of our minds as we are working with them. For example, it's not a matter of whether a service is or is not reusable; it's the degree of reusability that we can realize through its design that we are primarily concerned with.

Most of the chapters in this book explore specific measures to which a principle can be applied and further provide recommendations for how these levels can be classified and documented. Additional supporting practices are provided in Chapter 15.

**SUMMARY OF KEY POINTS**

- Design principles can be effectively realized by applying them as part of formal analysis and design processes.
- Design principles can be further applied consistently by incorporating them into official design standards.
- Every principle can be applied to a certain extent.

## 5.2 Principle Profiles

Each of the chapters in Part II contains a section that summarizes a design principle within a standard profile table. Provided here are brief descriptions of the fields within the standard profile table:

- *Short Definition*—A concise, single-statement definition that establishes the fundamental purpose of the principle.
- *Long Definition*—A longer description of the principle that provides more detail as to what it is intended to accomplish.
- *Goals*—A list of specific design goals that are expected from the application of the principle. Essentially, this list provides the ultimate results of the principle's realization.
- *Design Characteristics*—A list of specific design characteristics that can be realized via the application of the principle. This provides some insight as to how the principle ends up shaping the service.
- *Implementation Requirements*—A list of common prerequisites for effectively applying the design principle. These can range from technology to organizational requirements.
- *Web Service Region of Influence*—A simple diagram that highlights the regions within a physical Web service architecture affected by the application of the principle. The standard Web service representation (consisting of core service logic, messaging logic, and the service contract) is used repeatedly. Red shaded spheres indicate the areas of the Web service the principle is most likely to affect. The darker the shading, the stronger the potential influence.

Chapters are further supplemented with the following sections:

- *Abstract*—An introductory section that explains each design principle outside of the context of SOA. This is a helpful perspective in understanding how service-orientation positions design principles. The title of this section incorporates the name of the principle as follows: *[Principle Name] in Abstract.*
- *Origins*—A section that establishes the roots of a given principle by drawing from past architectures and design approaches. By understanding the history of each design principle, it becomes clear how service-orientation is truly an evolutionary paradigm. The format of this section's title is as follows: *Origins of [Principle Name].*

- *Levels*—As explained earlier in the *Apply Principles to a Feasible Extent* section, each principle can be realized to a certain degree. Most chapters provide suggested labels for categorizing the level to which a principle has been applied, primarily for measuring and communication purposes. The section title is structured as follows: *Levels of [Principle Name].*
- *Service Design*—Several chapters explore supplementary topics that highlight additional design considerations associated with a principle. These are found in a section called *[Principle Name] and Service Design*. (Note that the following *Service Models* and *Relationships* sections exist as sub-sections to the *Service Design* section.)
- *Granularity*—Whenever the application of a design principle raises issues or concerns regarding any of the four design granularity types (as explained in the upcoming *Principles and Design Granularity* section) a separate section entitled *[Principle Name] and Granularity* is added.
- *Service Models*—Where appropriate, a principle's influence on the design of each of the four primary service models (entity, utility, task, and orchestrated task) is described in a section titled *[Principle Name] and Service Models*.
- *Relationships*—To fully appreciate the dynamics behind service-orientation, an understanding of how the application of one principle can potentially affect others is required. Each chapter provides a section titled *How [Principle Name] Affects Other Principles* wherein inter-principle relationships are explored.
- *Risks*—Finally, every chapter dedicated to a design principle concludes with a list of risks associated with using or abstaining from the use of the principle. This list is provided in a section titled *Risks Associated with [Principle Name].*

Every effort was made to keep the format of the next eight chapters consistent so that aspects of individual principles can be effectively compared and contrasted.

**NOTE**

Principle profiles should not be confused with service profiles. The former represents a regular section format within the upcoming chapters, whereas the latter is a type of document for recording service meta details. Service profiles are described in Chapter 15.

**SUMMARY OF KEY POINTS**

- Each chapter summarizes a design principle using a standard profile section.
- Design principles are further documented with additional sections that explore various aspects of their origin and application.

## 5.3 Design Pattern References

The eight design principles in this book were documented in alignment with an SOA design pattern catalog published separately in the book *SOA: Design Patterns,* another title that is part of the *Prentice Hall Service-Oriented Computing Series from Thomas Erl.* This book expresses service-orientation through a fundamental pattern language and provides a collection of advanced design patterns for solving common problems.

Because these two books were written together, there is a strong correlation between the utilization of design principles and select design patterns that provide related solutions in support of realizing service-orientation. Fundamental design patterns are often tied directly to the design characteristics established by a particular design principle, whereas advanced design patterns more commonly solve problems that can be encountered when attempting to apply a principle under certain circumstances.

Throughout the chapters in Part II, references to related design patterns are provided. These references are further summarized in Appendix C.

## 5.4 Principles that Implement vs. Principles that Regulate

Before exploring the design principles individually, it is worth positioning them as they relate to the realization of physical service design characteristics. On a fundamental level we can group principles into two broad categories:

- Principles that primarily result in the implementation of specific service design characteristics.
- Principles that primarily shape and regulate the application of other principles.

The following principles fall into the first category:

- Standardized Service Contract
- Service Reusability

- Service Autonomy
- Service Statelessness
- Service Discoverability

As explained throughout Chapters 6, 9, 10, 11, and 12, the application of any one of these principles results in very specific design qualities. Some affect the service contract, while others are more focused on the underlying service logic. However, all result in the implementation of characteristics that shape the physical service design.

This leaves us with the remaining three that fall into the "regulatory" category:

- Service Loose Coupling
- Service Abstraction
- Service Composability

After studying Chapters 7, 8, and 13, it becomes evident that while these principles also introduce some new characteristics, they primarily influence how and to what extent the service design characteristics associated with other principles are implemented (Figure 5.1).

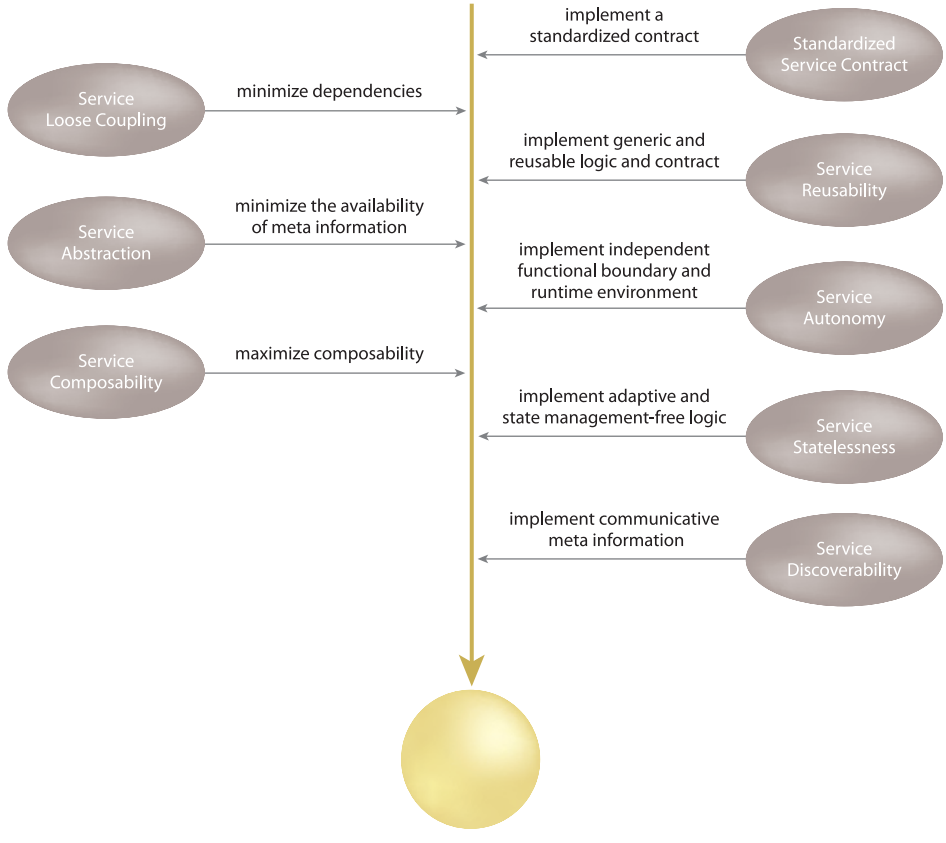



**Figure 5.1**

While the principles on the right-hand side want to add specific physical characteristics to the service design, the principles on the left act as regulators to ensure that these characteristics are implemented in a coordinated and appropriate manner.

Furthermore, each chapter explores how principles inter-relate. Specifically, the manner in which a design principle affects the application of others is documented. Figure 5.2, for example, provides an indication as to how two of the "regulatory" principles relate to each other.

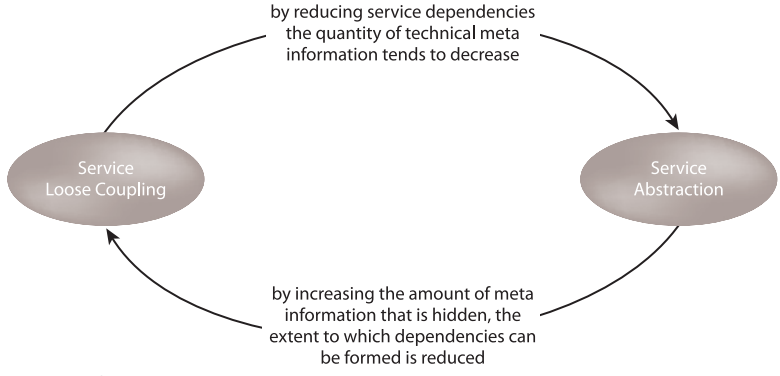



**Figure 5.2**
The Service Loose Coupling and Service Abstraction principles share a common dynamic in that the application of each supports the other.

**SUMMARY OF KEY POINTS**

- Five of the eight design principles establish concrete service design characteristics.
- The remaining three design principles also introduce design characteristics but act more as regulatory influences.

## 5.5 Principles and Service Implementation Mediums

Service logic can exist in different forms. It can be implemented as the core logic component within a Web service, as a standalone component with a public interface, or even within an event-driven service agent. The choice of implementation medium or format can be influenced by environmental constraints, architectural considerations, as well as the application of various design patterns.

Service-orientation design principles shape both service logic and service contracts. There is an emphasis on the Web service medium because it provides the most potential to apply key principles to the greatest extent. For example, contract-related principles may not apply as much to logic encapsulated within an event-driven service agent. This does not make the logic any less service-oriented; it only limits the principles that need to be taken into account during its development.

### "Capability" vs. "Operation" vs. "Method"

To support the on-going distinction between a service in abstract and a service implemented as a Web service, separate terms are used to refer to the functions a service can provide.

A *service capability* represents a specific function of a service through which the service can be invoked. As a result, service capabilities are expressed within the service contract. A service can have capabilities regardless of how it is implemented.

A *service operation* specifically refers to a capability within a service that is implemented as a Web service. Similarly, a *service method* represents a capability that is part of a service that exists as a component.

Note that as mentioned early on in Chapter 3, when the term "capability" is used in this book, it implicitly refers to capabilities expressed by the service contract. If there is a need to reference internal service capabilities that are not part of the contract, they will be explicitly qualified as such.

## 5.6 Principles and Design Granularity

The term "granularity" is most commonly used to communicate the level of (or absence of) detail associated with some aspect of software program design. Within the context of service design, we are primarily concerned with the granularity of the service contract and what it represents.

Within a service, different forms of granularity exist, all of which can be impacted by how service-orientation design principles are applied. The following sections document four specific types of design granularity, three of which are further referenced in Figure 5.3.

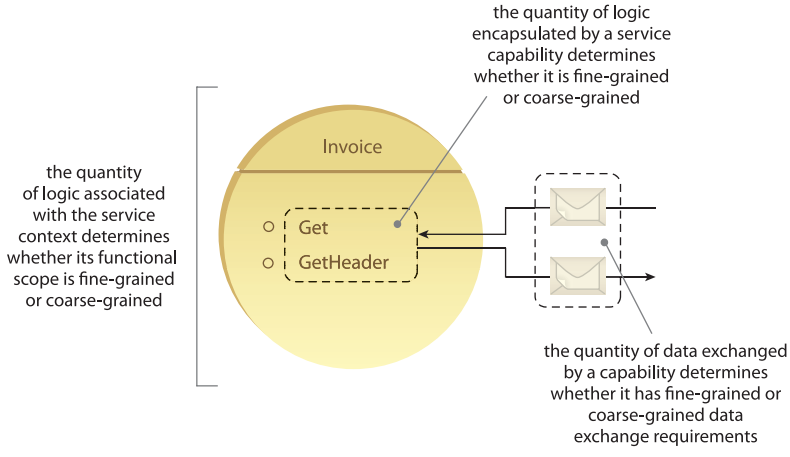



**Figure 5.3**

In this example, an Invoice entity service will tend to have a coarse-grained functional scope. However, it is exposing both coarse-grained (Get) and fine-grained (GetHeader) capabilities. Furthermore, because the GetHeader capability will return less data than the Get capability (which returns an entire invoice document), the GetHeader capability's data granularity is also considered fine.

### Service Granularity

The granularity of the service's functional scope, as determined by its functional context, is simply referred to as *service granularity*. A service's overall granularity does not reflect the amount of logic it currently encapsulates but instead the quantity of potential logic it could encapsulate, based on its context. A coarse-grained service, for example, would have a broad functional context, regardless of whether it initially expresses one or ten capabilities.

### Capability Granularity

*Capability granularity* represents the functional scope of a specific capability as it currently exists. As a rule of thumb, a fine-grained capability will have less work to do than a coarse-grained one.

### Data Granularity

The quantity of data a capability needs to exchange in order to carry out its function represents its level of *data granularity*. There has been a tendency for services implemented as Web services to exchange document-centric messages—messages containing entire information sets or business documents. Because the quantity of data is larger, this would be classified as coarse-grained data granularity.

Document-centric messages are in sharp contrast to traditional RPC-style communication, which typically relies on the exchange of smaller (fine-grained) amounts of parameter data.

### Constraint Granularity

The amount of detail with which a particular constraint is expressed is referred to as a measure of *constraint granularity*. The schema or data model representing the structure of the information being exchanged by a capability can define a series of specific validation constraints (data type, data length, data format, allowed values, etc.) for a given value. This would represent a fine-grained (detailed) constraint granularity for that value, as opposed to a coarse-grained level of constraint granularity that would permit a range of values with no predefined length or format restrictions, as represented by the first element definition in Example 5.1.

```
<xsd:element name="ProductCode" type="xsd:string"/>

<xsd:element name="ProductCode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="4"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="ProductCode">
<xsd:simpleType>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-9]{4}"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

**Example 5.1**

Three variations of the same XML schema element definition. The first is clearly a coarse-grained constraint because it allows the product code to exist as an open-ended string value. The second is less coarse-grained because it restricts the product code length to one to four characters. The last element is a fine-grained constraint because it dictates that the product code must be four characters and that each character be a number between 0 and 9.

Constraint granularity can be associated with individual parameters processed by a capability or with the capability as a whole. For example, the same capability may accept

a body of input data comprised of two separate values, one of which is subject to fine-grained constraints and the other of which is validated against a coarse-grained constraint. The three code samples in Example 5.1 could alternatively exist as different types with different names but with the same constraints and as part of the same service capability (or Web service operation).

It is also important to note that constraint granularity is generally measured in relation to the validation logic present in the service contract only. This measure therefore excludes validation constraints that may be applied by the underlying service logic. Whereas a capability defined within a contract may have coarse constraint granularity, the actual capability logic may apply more fine-grained constraints after input values have been validated against the service contract.

**NOTE**

There are no rules about how forms of granularity can be combined. For example, it would not be uncommon for a coarse-grained service to provide fine-grained capabilities that exchange coarse-grained data validated against fine-grained constraints.

### Sections on Granularity Levels

There is no one principle that dictates granularity levels for a service design. Instead, several service-orientation principles impact the various types of granularity in different ways. Those chapters that cover principles affecting design granularity typically address this issue within the standard *[Principle Name] and Service Design* section.

**SUMMARY OF KEY POINTS**

- Service granularity refers to the functional scope of the service as a whole, as defined by its functional context.
- Capability granularity refers to the functional scope of a specific capability.
- Data granularity refers to the volume of data exchanged by a service capability.
- Constraint granularity refers to the level of detail to which validation logic is defined for a particular parameter or capability within the service contract.

## 5.7 CASE STUDY BACKGROUND

Each of the upcoming eight chapters concludes with a case study example that demonstrates the application of a principle. Specifically, a service delivery project underway at Cutit Saws forms the basis for these examples, as a modest set of services are developed to automate the Lab Project business process.

Up next is a description of this process that will help establish some overall context. Note, however, that the focus of subsequent case study examples is not on the nature of the business process logic but more so on the design issues pertaining to the incorporation of service-orientation principles.

### The Lab Project Business Process

The following is a highly simplified version of a lab project in which the assembly of materials and the application of predefined (and sometimes newly created) formulas undergoes a series of verification checks and then a final simulation. Note that in order to preserve clarity surrounding the flow of the process logic, regular industry terms and chemistry-related terminology is intentionally avoided.

A lab project, within the context of this solution, is the equivalent of a simulated experiment. Using a customized user-interface, a lab technician assembles a combination of ingredients (purchased and/or developed materials) and retrieves either one or more existing base formulas or creates one or more newly developed base formulas. A base formula is essentially a documentation of existing compounds (previous mixtures of ingredients or elements).

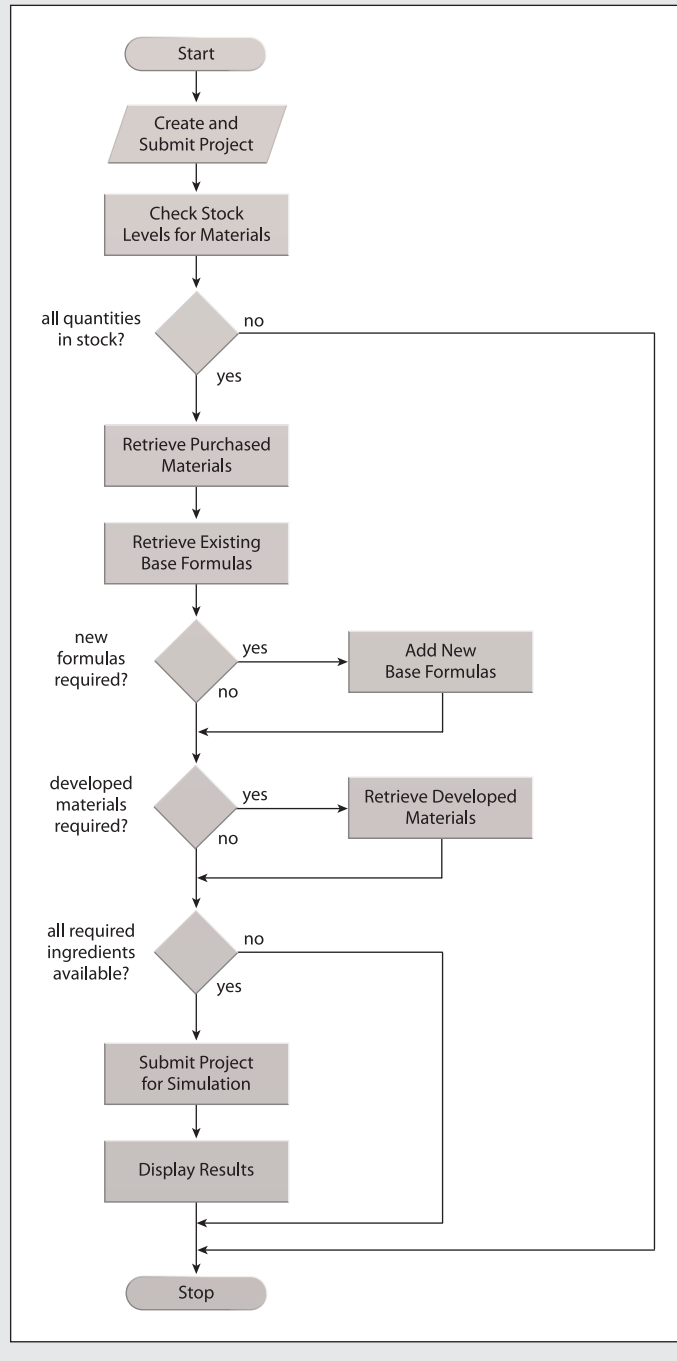
Once all of the information is in place, the lab project is executed ("run"), and the solution retrieves the required information, as per the process description that follows. If all of the needed ingredients are available, the solution interacts with a simulator program to graphically display the results of the experiment. If any ingredients are missing or certain formula combinations are not possible, the solution will reject the experiment configuration and terminate the project.

Here are descriptions for the primary process steps, which are further displayed in the workflow diagram in Figure 5.4.

1. Issue a stock level check on all required materials. If any stock levels are lower than the requested quantities, terminate the process.
2. Retrieve information about required purchased materials. This can include lab equipment, tools, and disposable materials (gloves, swabs, etc.) in addition to materials used as ingredients for experiments.
3. Retrieve a list of the requested base formulas, filtered using criteria pertinent to the current project.
4. If a new base formula is being added, generate the base formula record and add the formula to the base formulas list for this project.
5. If developed materials are required, retrieve their corresponding data. This step is performed after the base formulas are defined to ensure that all required ingredients are accounted for.
6. Perform a validation check to ensure that all purchased and developed ingredients are available in order for the defined formulas to be applied.
7. Submit the collected data to the simulator.
8. Output the results in a predefined report format.

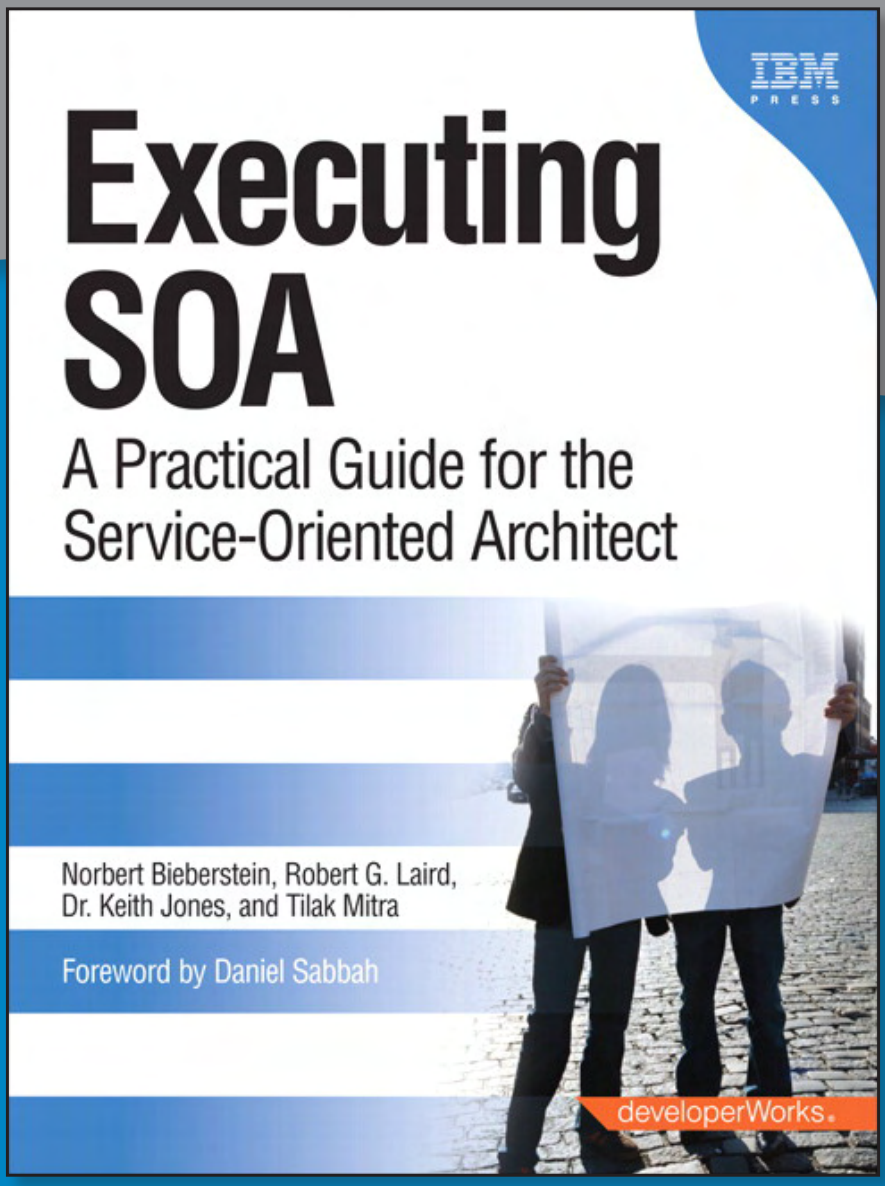
Note that the process contains one additional step that has been excluded from the preceding description and workflow diagram. If the simulation attempt fails, the returned report contains error information, and a separate sub-process is invoked that contains compensation steps including notification. This sub-process is only somewhat relevant to the case study example in Chapter 10 where processing subsequent to the completion of the report generation is referenced.

**Figure 5.4**
The workflow logic for the Lab Project business process.



**NOTE**

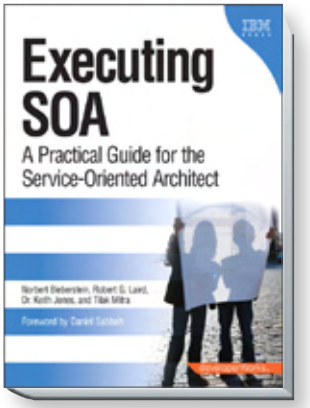
Just a reminder that, as explained in the *What this Book Does Not Cover* section of Chapter 1, the focus of this book and the upcoming chapters in Part II is on the design of *services* for SOA, not the design of SOA itself. Architectural design issues are addressed separately as part of the book *SOA: Design Patterns*.




BUY ME

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

BUY ME




# Executing SOA

## A Practical Guide for the Service-Oriented Architect

### The Expert, Practical Guide to Succeeding with SOA in the Enterprise

In *Executing SOA*, four experienced SOA implementers share realistic, proven, "from-the-trenches" guidance for successfully delivering on even the largest and most complex SOA initiative.

This book follows up where the authors' best-selling Service-Oriented Architecture Compass left off, showing how to overcome key obstacles to successful SOA implementation and identifying best practices for all facets of execution—technical, organizational, and human. Among the issues it addresses: introducing a services discipline that supports collaboration and information process sharing; integrating services with preexisting technology assets and strategies; choosing the right roles for new tools; shifting culture, governance, and architecture; and bringing greater agility to the entire organizational lifecycle, not just isolated projects.

*Executing SOA* is an indispensable resource for every enterprise architect, technical manager, and IT leader tasked with driving value from SOA in complex environments.

COVERAGE INCLUDES

- Implementing SOA governance that reflects the organization's strategic and business focus
- Running SOA projects successfully: practical guidelines and proven methodologies around service modeling and design
- Leveraging reusable assets: making the most of your SOA repository
- Enabling the architect to choose the correct tools and products containing the features required to execute on the SOA method for service design and implementation
- Defining information services to get the right information to the right people at the right time
- Integrating SOA with Web 2.0 and other innovative products and solutions
- Providing highly usable human interfaces in SOA environments

AVAILABLE
- BOOK: 9780132353748
- SAFARI ONLINE Safari Books Online
- EBOOK: 0137149786
- KINDLE: B0019HW0JC

## 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. LAIRDIS 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™

ibmpressbooks.com

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.

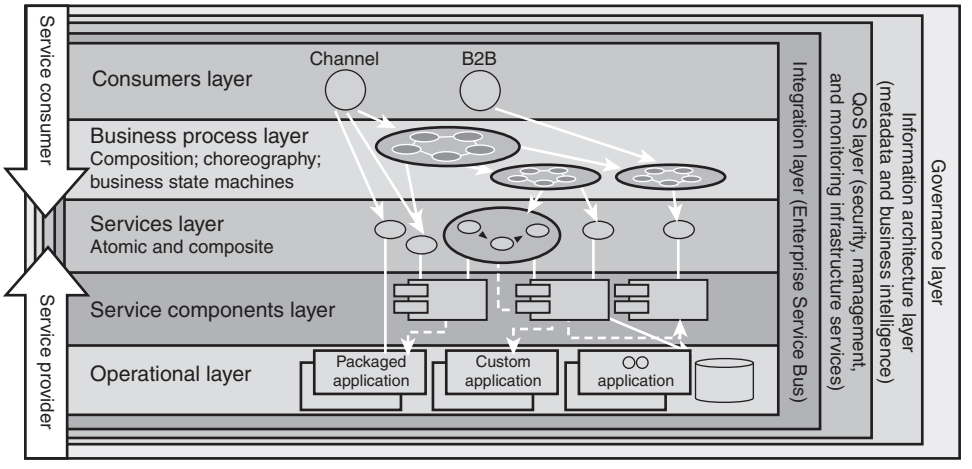



**Figure 4-1** Logical view of SOA Reference Architecture

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.

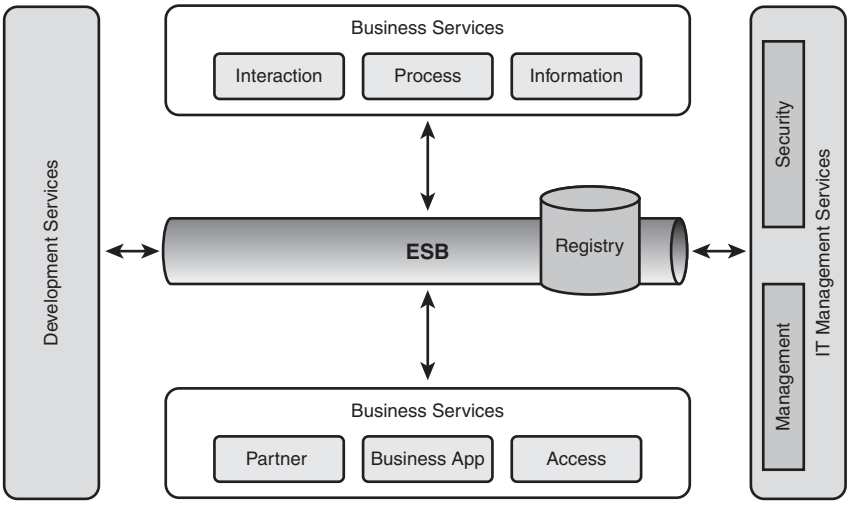



**Figure 4-2** Logical view of an ESB in the integration layer

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.

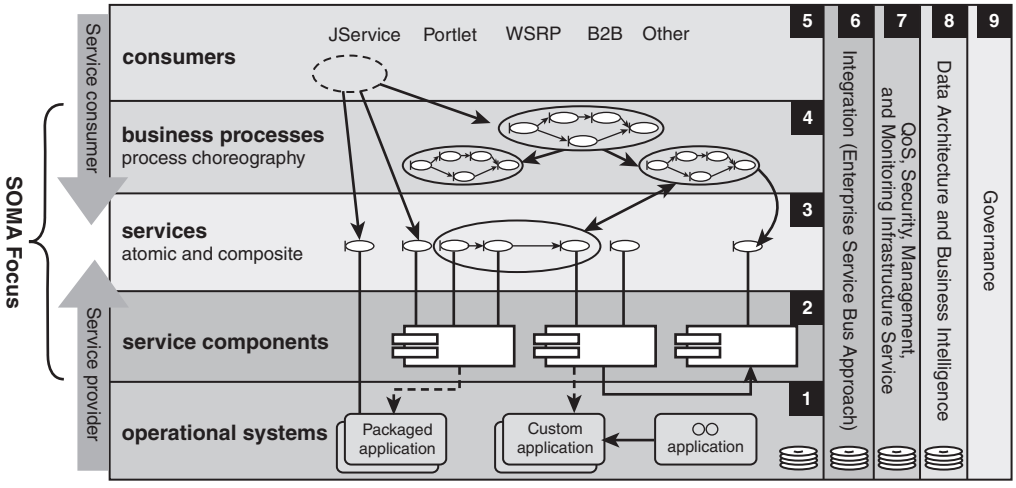



**Figure 4-3** The focus of SOMA is on Layer 2 through Layer 4.

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.

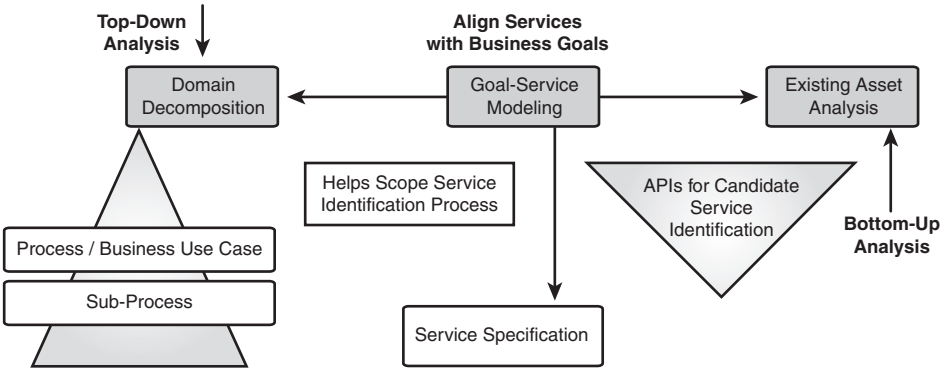



**Figure 4-4** The three different techniques for service identification

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.

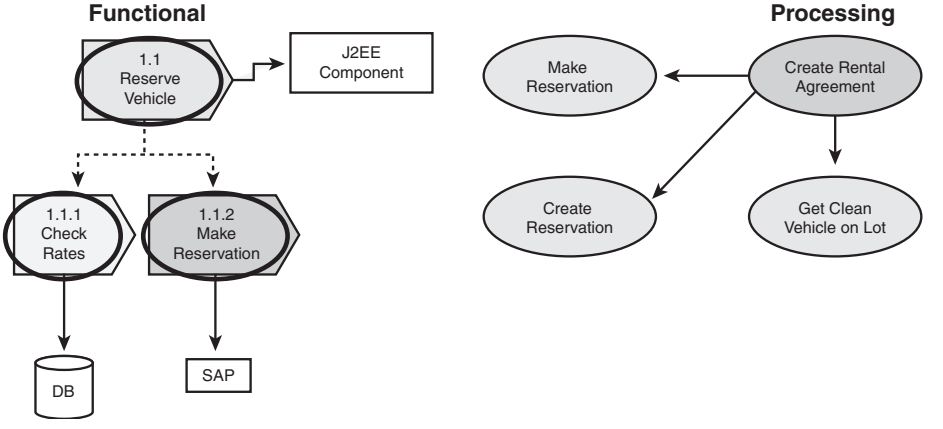



**Figure 4-5** The two general types of service dependencies

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.

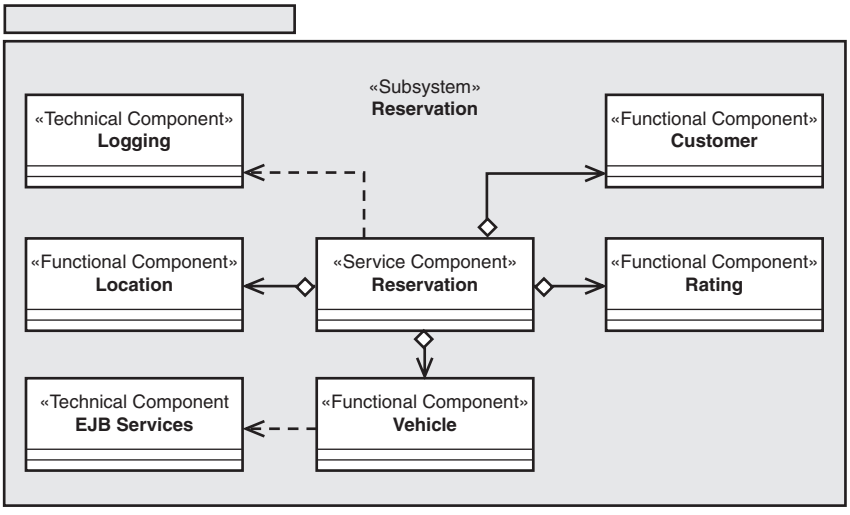



**Figure 4-6** The relationship between a subsystem and its constituent service, functional, and technical components

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.

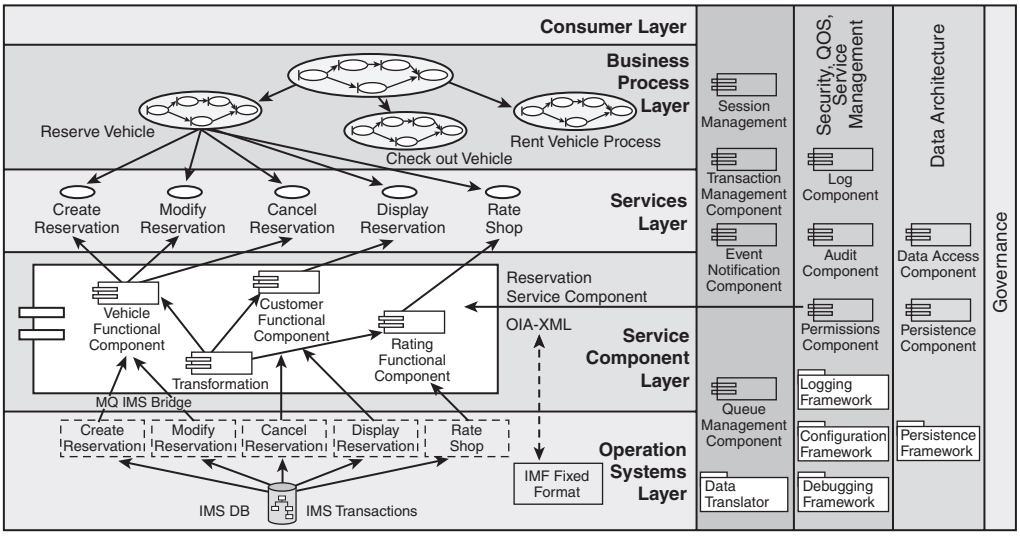



**Figure 4-7** Allocation of software artifacts on the layers of the SOA reference architecture

### 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/

Thomas Erl, Anish Karmarkar,
Priscilla Walmsley, Hugo Haas,
L. Umit Yalcinalp Ph.D.,
Kevin Liu, David Orchard,
Andre Tost, and James Pasley

BUY ME


# Web Service Contract Design and Versioning for SOA

UNDERSTANDING WEB SERVICE CONTRACT TECHNOLOGIES: Initial chapters and ongoing supplementary content help even the most inexperienced professional get up to speed on how all of the different technologies and design considerations relate to the creation of Web service contracts.

FUNDAMENTAL AND ADVANCED WSDL: Tutorial coverage of WSDL 1.1 and 2.0 and detailed descriptions of their differences is followed by numerous advanced WSDL topics and design techniques.

FUNDAMENTAL AND ADVANCED XML SCHEMA: XML Schema basics are covered within the context of Web services and SOA, after which advanced XML Schema chapters delve into a variety of specialized message design considerations and techniques.

FUNDAMENTAL AND ADVANCED WS-POLICY: Topics, such as Policy Expression Structure, Composite Policies, Operator Composition Rules, and Policy Attachment establish a foundation upon which more advanced topics, such as policy reusability and centralization, nested, parameterized, and ignorable assertions are covered.

FUNDAMENTAL MESSAGE DESIGN WITH SOAP: A broad range of message design-related topics are covered, including SOAP message structures, SOAP nodes and roles, SOAP faults, designing custom SOAP headers and working with industry-standard SOAP headers.

ADVANCED MESSAGE DESIGN WITH WS-ADDRESSING: The art of message design is taken to a new level with in-depth descriptions of WS-Addressing endpoint references (EPRs) and MAP headers and an exploration of how they are applied via SOA design patterns.

ADVANCED MESSAGE DESIGN WITH MTOM, AND SWA: Developing SOAP messages capable of transporting large documents or binary content is explored with a documentation of the MTOM packaging and serialization framework, together with the SOAP with Attachments (SwA) standard and the related WS-I Attachments Profile.

VERSIONING TECHNIQUES AND STRATEGIES: Fundamental versioning theory starts off a series of chapters that dive into a variety of versioning techniques based on proven SOA design patterns.

WEB SERVICE CONTRACTS AND SOA: The constant focus of this book is on the design and versioning of Web service contracts in support of SOA and service-orientation. Relevant SOA design principles and design patterns are periodically discussed to demonstrate how specific Web service technologies can be applied and further optimized.

AVAILABLE
- BOOK: 9780136135173
- SAFARI ONLINE
- EBOOK: 0131360612
- KINDLE: B001FA0MAW

**The Ultimate Guide for Designing and Governing Web Service Contracts**

PRENTICE HALL

informit.com/ph

# Chapter 3

# SOA Fundamentals and Web Service Contracts

As we established in Chapter 1, this is not a book about SOA, nor is it a book about Web services. Several books have been published over the past few years that address these topics individually. What we are focused on is the design and versioning of Web service contracts in support of service-oriented solution design. What this requires us to do is explore the marriage of Web service contract technologies with the service-orientation design paradigm and the service-oriented architectural model.

As a starting point, we first need to establish some fundamental terms and concepts associated with service-oriented computing and with an emphasis of how these terms and concepts relate to Web service contracts.

## 3.1 Basic SOA Terminology

This section borrows some content from SOAGlossary.com to provide the following term definitions:

- Service-Oriented Computing
- Service-Orientation
- Service-Oriented Architecture (SOA)
- Service
- Service Models
- Service Composition
- Service Inventory
- Service-Oriented Analysis
- Service Candidate
- Service-Oriented Design
- Web Service
- Service Contract
- Service-Related Granularity

If you are already an experienced SOA professional, then you might want to just skim through this part of the book. The defined terms are used here and there throughout subsequent chapters.

### Service-Oriented Computing

*Service-oriented computing* is an umbrella term that represents a new generation distributed computing platform. As such, it encompasses many things, including its own design paradigm and design principles, design pattern catalogs, pattern languages, a distinct architectural model, and related concepts, technologies, and frameworks.

Service-oriented computing builds upon past distributed computing platforms and adds new design layers, governance considerations, and a vast set of preferred implementation technologies, many of which are based on the Web services framework.

In this book we refer primarily to the strategic goals of service-oriented computing as they tie into approaches for Web service contract design and versioning. These goals are briefly described in the *Service-Oriented Computing Goals and Web Service Contracts* section.

### Service-Orientation

*Service-orientation* is a design paradigm intended for the creation of solution logic units that are individually shaped so that they can be collectively and repeatedly utilized in support of the realization of the specific strategic goals and benefits associated with SOA and service-oriented computing.

Solution logic designed in accordance with service-orientation can be qualified with "service-oriented," and units of service-oriented solution logic are referred to as "services." As a design paradigm for distributed computing, service-orientation can be compared to object-orientation (or object-oriented design). Service-orientation, in fact, has many roots in object-orientation and has also been influenced by other industry developments, including EAI, BPM, and Web services.

The service-orientation design paradigm is primarily comprised of eight specific design principles, as explained in the *Service-Orientation and Web Service Contracts* section. Several of these principles can affect the design of Web service contracts.

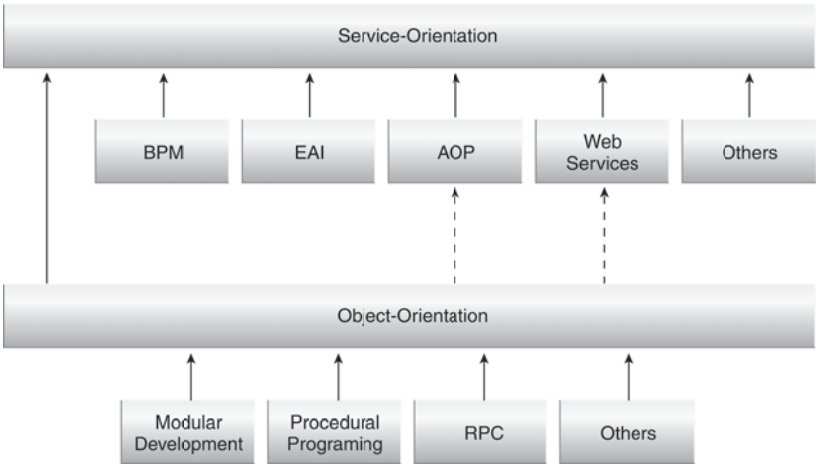



**Figure 3.1**

Service-orientation is an evolutionary design paradigm that owes much of its existence to established design practices and technology platforms.

### Service-Oriented Architecture (SOA)

*Service-oriented architecture* represents an architectural model that aims to enhance the agility and cost-effectiveness of an enterprise while reducing the overall burden of IT on an organization. It accomplishes this by positioning services as the primary means through which solution logic is represented. SOA supports service-orientation in the realization of the strategic goals associated with service-oriented computing.

As a form of technology architecture, an SOA implementation can consist of a combination of technologies, products, APIs, supporting infrastructure extensions, and various other parts. The actual complexion of a deployed service-oriented architecture is unique within each enterprise; however it is typified by the introduction of new technologies and platforms that specifically support the creation, execution, and evolution of service-oriented solutions. As a result, building a technology architecture around the service-oriented architectural model establishes an environment suitable for solution logic that has been designed in compliance with service-orientation design principles.

**NOTE**

Historically, the term "service-oriented architecture" (or "SOA") has been used so broadly by the media and within vendor marketing literature that it has almost become synonymous with service-oriented computing itself.

Note that the following service-oriented architecture types exist:

- Service Architecture
- Service Composition Architecture
- Service Inventory Architecture
- Service-Oriented Enterprise Architecture

As you may have guessed, we are primarily focused on the service architecture in this book. However, the decisions we make regarding the design of Web service contracts will ultimately affect the quality of related composition and inventory architectures.

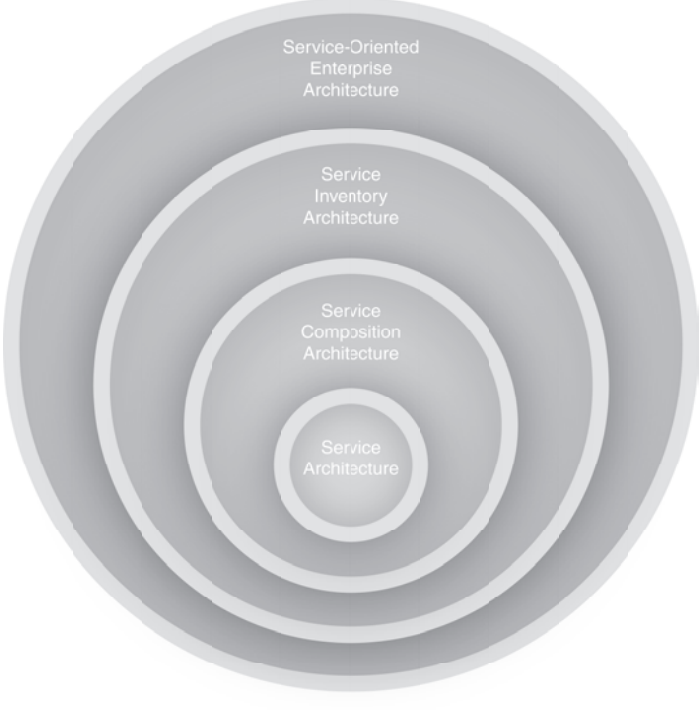



**Figure 3.2**
The layered SOA model that reveals how service-oriented architecture types can encompass each other. (These different architectural types are explained in Chapter 4 of the book *SOA Design Patterns*.)

### Service

A *service* is a unit of solution logic to which service-orientation has been applied to a meaningful extent. It is the application of service-orientation design principles that distinguishes a unit of logic as a service compared to units of logic that may exist solely as objects or components.

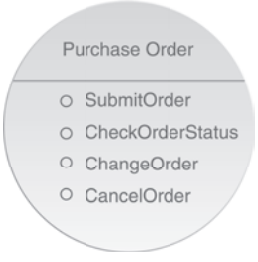



**Figure 3.3**
The chorded circle symbol is used to represent a service, primarily from a contract perspective.

Subsequent to conceptual service modeling, service-oriented design and development stages implement a service as a physically independent software program with specific design characteristics that support the attainment of the strategic goals associated with service-oriented computing.

Each service is assigned its own distinct functional context and is comprised of a set of capabilities related to this context. Therefore, a service can be considered a container of capabilities associated with a common purpose (or functional context). Capabilities are expressed in the service contract (defined shortly).

As we established earlier, this book is dedicated to the design of technical contracts for services built as Web services. Within a Web service contract, service capabilities are referred to as service *operations*.

### Service Models

A *service model* is a classification used to indicate that a service belongs to one of several predefined types based on the nature of the logic it encapsulates, the reuse potential of this logic, and how the service may relate to domains within its enterprise.

The following three service models are common to most enterprise environments and therefore common to most SOA projects:

- *Task Service* – A service with a non-agnostic functional context that generally corresponds to single-purpose, parent business process logic. A task service will usually encapsulate the composition logic required to compose several other services in order to complete its task.
- *Entity Service* – A reusable service with an agnostic functional context associated with one or more related business entities (such as invoice, customer, claim, etc.). For example, a Purchase Order service has a functional context associated with the processing of purchase order-related data and logic. Chapter 13 has a section dedicated to Web service contract design for entity services.

- *Utility Service* – Also a reusable service with an agnostic functional context, but this type of service is intentionally not derived from business analysis specifications and models. It encapsulates low-level technology-centric functions, such as notification, logging, and security processing.

Service models play an important role during service-oriented analysis and service-oriented design phases. Although the just listed service models are well established, it is not uncommon for an organization to create its own service models. Often these new classifications tend to be derived from one of the aforementioned fundamental service models.

**NOTE**

Most of the service contract examples in this book are for entity services that are required to deal with core business-related processing and the exchange of business documents.

#### *Agnostic Logic and Non-Agnostic Logic*

The term "agnostic" originated from Greek and means "without knowledge." Therefore, logic that is sufficiently generic so that it is not specific to (has no knowledge of) a particular parent task is classified as *agnostic* logic. Because knowledge specific to single purpose tasks is intentionally omitted, agnostic logic is considered multi-purpose. On the flipside, logic that is specific to (contains knowledge of) a single-purpose task is labeled as *non-agnostic* logic.

Another way of thinking about agnostic and non-agnostic logic is to focus on the extent to which the logic can be repurposed. Because agnostic logic is expected to be multi-purpose, it is subject to the Service Reusability principle with the intention of turning it into highly reusable logic. Once reusable, this logic is truly multi-purpose in that it, as a single software program (or service), can be used to automate multiple business processes.

Non-agnostic logic does not have these types of expectations. It is deliberately designed as a single-purpose software program (or service) and therefore has different characteristics and requirements.

### Service Composition

A *service composition* is an aggregate of services collectively composed to automate a particular task or business process. To qualify as a composition, at least two participating services plus one composition initiator need to be present. Otherwise, the service interaction only represents a point-to-point exchange.

Service compositions can be classified into primitive and complex variations. In early service-oriented solutions, simple logic was generally implemented via point-to-point exchanges or primitive compositions. As the surrounding technology matured, complex compositions became more common.

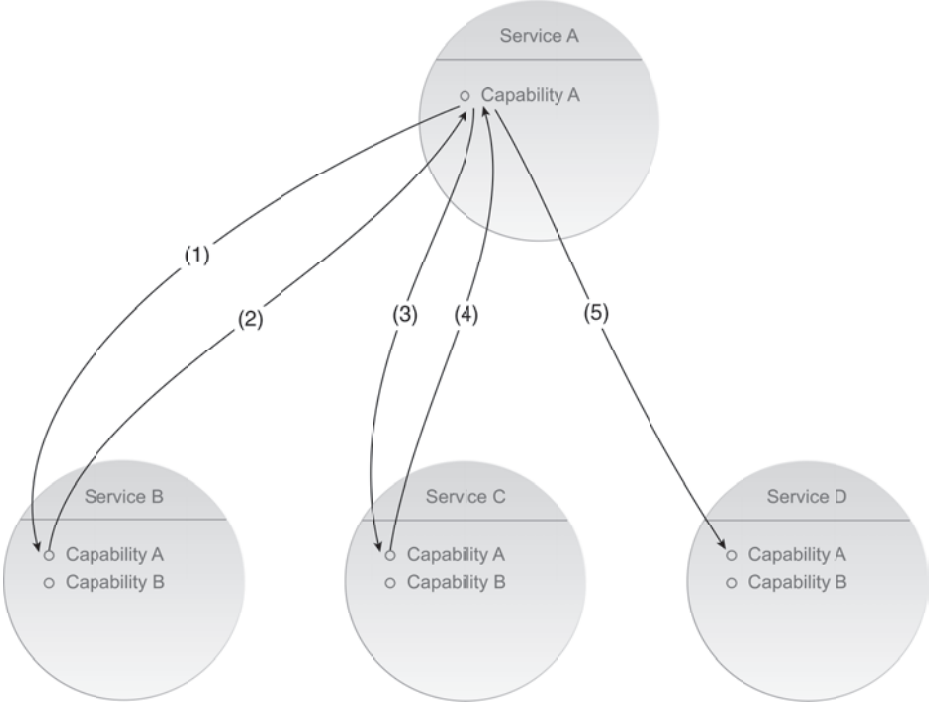



**Figure 3.4**
A service composition comprised of four services. The arrows indicate a sequence of modeled message exchanges. Note arrow #5 representing a one-way, asynchronous data delivery from Service A to Service D.

Much of the service-orientation design paradigm revolves around preparing services for effective participation in numerous complex compositions—so much so that the Service Composability design principle exists, dedicated solely to ensuring that services are designed in support of repeatable composition.

How service contracts are designed will influence the effectiveness and complexity potential of service compositions. Various contract-related granularity levels will determine the quantity of runtime processing and data exchange required—qualities that can end up hindering or enabling composition performance. Furthermore, techniques, such as those provided by the Contract Denormalization and Concurrent Contracts patterns,

can help optimize composition designs. (These design patterns are briefly explained at the end of this chapter in the *SOA Design Patterns and Web Service Contracts* section.)

### Service Inventory

A *service inventory* is an independently standardized and governed collection of complementary services within a boundary that represents an enterprise or a meaningful segment of an enterprise. When an organization has multiple service inventories, this term is further qualified as *domain service inventory*.

Service inventories are typically created through top-down delivery processes that result in the definition of *service inventory blueprints*. The subsequent application of service-orientation design principles and custom design standards throughout a service inventory is of paramount importance so as to establish a high degree of native inter-service interoperability. This supports the repeated creation of effective service compositions in response to new and changing business requirements.

It is worth noting that the application of the Standardized Service Contract principle is intended to be limited to the boundary of a service inventory, as are design standards-related patterns, such as Canonical Transport and Canonical Schema.

### Service-Oriented Analysis

*Service-oriented analysis* represents one of the early stages in an SOA initiative and the first phase in the service delivery cycle. It is a process that begins with preparatory information gathering steps that are completed in support of a service modeling sub-process that results in the creation of conceptual service candidates, service capability candidates, and service composition candidates.

The service-oriented analysis process is commonly carried out iteratively, once for each business process. Typically, the delivery of a service inventory determines a scope that represents a meaningful domain or the enterprise as a whole. All iterations of a service-oriented analysis then pertain to that scope, with an end-result of a service inventory blueprint.

**NOTE**

Visit SOAMethodology.com for an explanation of the iterative service-oriented analysis process.

A key success factor of the service-oriented analysis process is the hands-on collaboration of both business analysts and technology architects. The former group is especially involved in the definition of service candidates with a business-centric functional context because they understand the business processes used as input for the analysis and because service-orientation aims to align business and IT more closely.

### Service Candidate

When conceptualizing services during the service modeling sub-process of the service-oriented analysis phase, services are defined on a preliminary basis and still subject to a great deal of change and refinement before they are handed over to the service-oriented design project stage responsible for producing physical service contracts.

The term "service candidate" is used to help distinguish a conceptualized service from an actual implemented service. You'll notice a few references to service candidates in this book, especially in some of the early case study content.

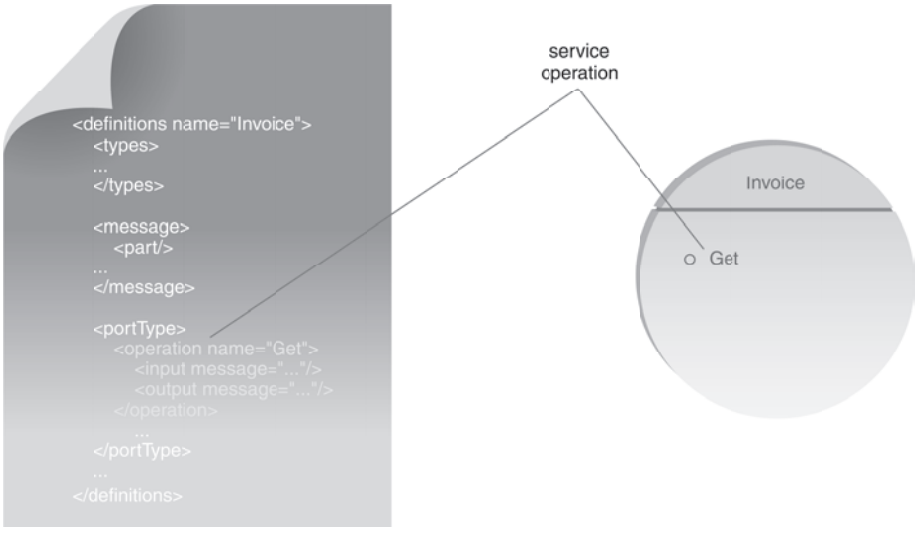



**Figure 3.5**

The chorded circle symbol (right) provides a simple representation of a service contract during both modeling and design stages. The Get operation (right) is first modeled and then forms the basis of the actual operation definition within a WSDL document (left).

### Service-Oriented Design

The *service-oriented design* phase represents a service delivery lifecycle stage dedicated to producing service contracts in support of the well-established "contract-first" approach to software development.

The typical starting point for the service-oriented design process is a service candidate that was produced as a result of completing all required iterations of the service-oriented analysis process. Service-oriented design subjects this service candidate to additional considerations that shape it into a technical service contract in alignment with other service contracts being produced for the same service inventory.

There is a different service-oriented design process for each of the three common service models (task, entity, and utility). The variations in process steps primarily accommodate different priorities and the nature of the logic being expressed by the contract.

This book does not discuss the process of service-oriented design in detail, but there are references to some of the considerations raised by the "contract-first" emphasis of this process.

### Web Service

A *Web service* is a body of solution logic that provides a physically decoupled technical contract consisting of a WSDL definition and one or more XML Schema and/or WS-Policy definitions. As we will explore in this book, these documents can exist in one physical file or be spread across multiple files and still be part of one service contract. Spreading them out makes them reusable across multiple contracts.

The Web service contract exposes public capabilities as operations, establishing a technical interface comparable to a traditional application programming interface (API) but without any ties to proprietary communications framework.

The logic encapsulated by a Web service does not need to be customized or component-based. Legacy application logic, for example, can be exposed via Web service contracts through the use of service adapter products. When custom-developed, Web service logic is typically based on modern component technologies, such as Java and .NET. In this case, the components are further qualified as core service logic.

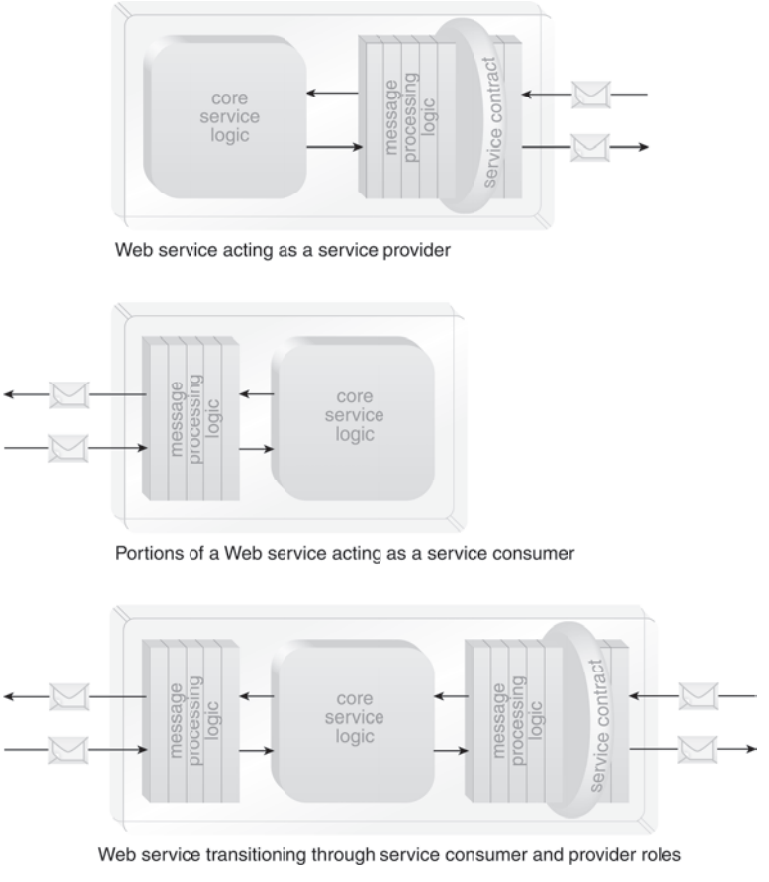



**Figure 3.6**

Three variations of a single Web service showing the different physical parts of its architecture that come into play, depending on the role it assumes at runtime. Note the cookie-shaped symbol that represents the service contract wedged in between layers of agent-driven message processing logic. This is the same chorded circle symbol shown earlier but from a different perspective.

## Service Contract

A *service contract* is comprised of one or more published documents that express meta information about a service. The fundamental part of a service contract consists of the documents that express its technical interface. These form the technical service contract, which essentially establishes an API into the functionality offered by the service via its capabilities.

When services are implemented as Web services, the most common service description documents are the WSDL definition, XML schema definition, and WS-Policy definition. A Web service generally has one WSDL definition, which can link to multiple XML schema and policy definitions. When services are implemented as components, the technical service contract is comprised of a technology-specific API.

A service contract can be further comprised of human-readable documents, such as a Service Level Agreement (SLA) that describes additional quality-of-service features, behaviors, and limitations. As we discuss in the WS-Policy chapters, several SLA-related requirements can also be expressed in machine-readable format as policies.

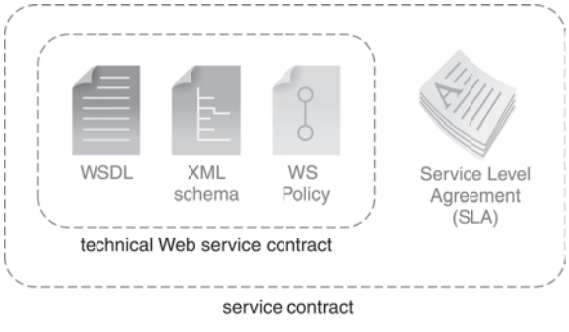



**Figure 3.7**
The common documents that comprise the technical Web service contract, plus a human-readable SLA.

Within service-orientation, the design of the service contract is of paramount importance—so much so, that the Standardized Service Contract design principle and the aforementioned service-oriented design process are dedicated solely to the standardized creation of service contracts.

Note that because this book is focused only on technical contracts for Web services, the terms "service contract" and "Web service contract" are used interchangeably.

### Service-Related Granularity

When designing services, there are different granularity levels that need to be taken into consideration, as follows:

- *Service Granularity* – Represents the functional scope of a service. For example, fine-grained service granularity indicates that there is little logic associated with the service's overall functional context.
- *Capability Granularity* – The functional scope of individual service capabilities (operations) is represented by this granularity level. For example, a GetDetail

capability will tend to have a finer measure of granularity than a GetDocument capability.

- *Constraint Granularity* – The level of validation logic detail is measured by constraint granularity. The more coarse constraint granularity is, the less constraints (or smaller the amount of validation logic) a given capability will have.
- *Data Granularity* – This granularity level represents the quantity of data processed. From a Web service perspective, this corresponds to input, output, and fault messages. A fine level of data granularity is equivalent to a small amount of data.

Because the level of service granularity determines the functional scope of a service, it is usually determined during analysis and modeling stages that precede service contract design. Once a service's functional scope has been established, the other granularity types come into play and affect both the modeling and physical design of a Web service contract.

In this book you will especially notice references to constraint granularity because so much of contract design relates to the definition of validation logic constraints.

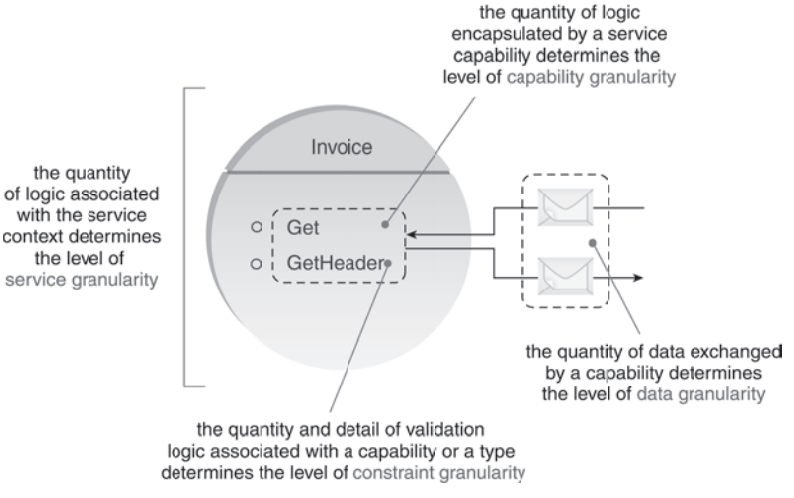



**Figure 3.8**

The four granularity levels that represent various characteristics of a service and its contract. Note that these granularity types are, for the most part, independent of each other.

### Further Reading

As mentioned at the beginning of this section, all of these terms are defined at SOA-Glossary.com. More detailed explanations are available at WhatIsSOA.com and in Chapters 3 and 4 of *SOA: Principles of Service Design*. If you are not familiar with service-oriented computing, it is recommended that you read through these additional descriptions.

## 3.2 Service-Oriented Computing Goals and Web Service Contracts

It's always good to get an idea of the big picture before diving into the details of any technology-centric topic. For this reason, we'll take the time to briefly mention the overarching goals and benefits associated with service-oriented computing as they relate to Web service contract design.

Because these goals are strategic in nature, they are focused on long-term benefit—a consideration that ties into both the design and governance of services and their contracts. An understanding of these long-term benefits helps provide a strategic context for many of the suggested techniques and practices in this guide.

Here's the basic list of the goals and benefits of service-oriented computing:

- Increased Intrinsic Interoperability
- Increased Federation
- Increased Vendor Diversification Options
- Increased Business and Technology Domain Alignment
- Increased ROI
- Increased Organizational Agility
- Reduced IT Burden

Although it might not be evident, service contract design touches each of these goals to some extent.

Let's explore how.

### Increased Intrinsic Interoperability

For services to attain a meaningful level of intrinsic interoperability, their technical contracts must be highly standardized and designed consistently to share common expressions and data models. This fundamental requirement is why project teams often must take control of their Web service contracts instead of allowing them to be auto-generated and derived from different sources.

### Increased Federation

Service-oriented computing aims to achieve a federated service endpoint layer. It is the service contracts that are the endpoints in this layer, and it is only through their consistent and standardized design that federation can be achieved. This, again, is a goal that is supported by the ability of a project team to customize and refine Web service contracts so that they establish consistent endpoints within a given service inventory boundary.

### Increased Vendor Diversification Options

For a service-oriented architecture to allow on-going vendor diversification, individual services must effectively abstract proprietary characteristics of their underlying vendor technology. The contract remains the only part of a service that is published and available to consumers. It must therefore be deliberately designed to express service capabilities without any vendor-specific details. This extent of abstraction allows service owners to extend or replace vendor technology. Vendor diversification is especially attainable through the use of Web services, due to the fact that they are supported by all primary vendors while providing a non-proprietary communications framework.

### Increased Business and Technology Domain Alignment

The service layers that tend to yield the greatest gains for service-oriented environments are those comprised of business-centric services (such as task and entity services). These types of services introduce an opportunity to effectively express various forms of business logic in close alignment with how this logic is modeled and maintained by business analysts.

This expression is accomplished through service contracts and it is considered so important that entire modeling processes and approaches exist to first produce a conceptual version of the service contract prior to its physical design.

### Strategic Benefits

The latter three goals listed in the previous bullet list represent strategic benefits that are achieved when attaining the first four goals. We therefore don't need to map the relevance of service contracts to each of them individually.

If we take the time to understand how central service contract design is to the ultimate target state we hope to achieve with service-oriented computing in general, it's clear to see why this book was written.

### Further Reading

Formal descriptions for each of these strategic goals are available at WhatIsSOA.com and in Chapter 3 of *SOA: Principles of Service Design*. While it's good to have an understanding of these goals and benefits, it is not required to learn the technologies covered in this book.

## 3.3 Service-Orientation and Web Service Contracts

To understand SOA is to understand service-orientation, the design paradigm that establishes what is required in order to create software programs that are truly service-oriented.

Service-orientation represents a design approach comprised of eight specific design principles. Service contracts tie into most but not all of these principles. Let's first introduce their official definitions:

- *Standardized Service Contract* – "Services within the same service inventory are in compliance with the same contract design standards."
- *Service Loose Coupling* – "Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment."
- *Service Abstraction* – "Service contracts only contain essential information and information about services is limited to what is published in service contracts."
- *Service Reusability* – "Services contain and express agnostic logic and can be positioned as reusable enterprise resources."
- *Service Autonomy* – "Services exercise a high level of control over their underlying runtime execution environment."

- *Service Statelessness* – "Services minimize resource consumption by deferring the management of state information when necessary."
- *Service Discoverability* – "Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted."
- *Service Composability* – "Services are effective composition participants, regardless of the size and complexity of the composition."

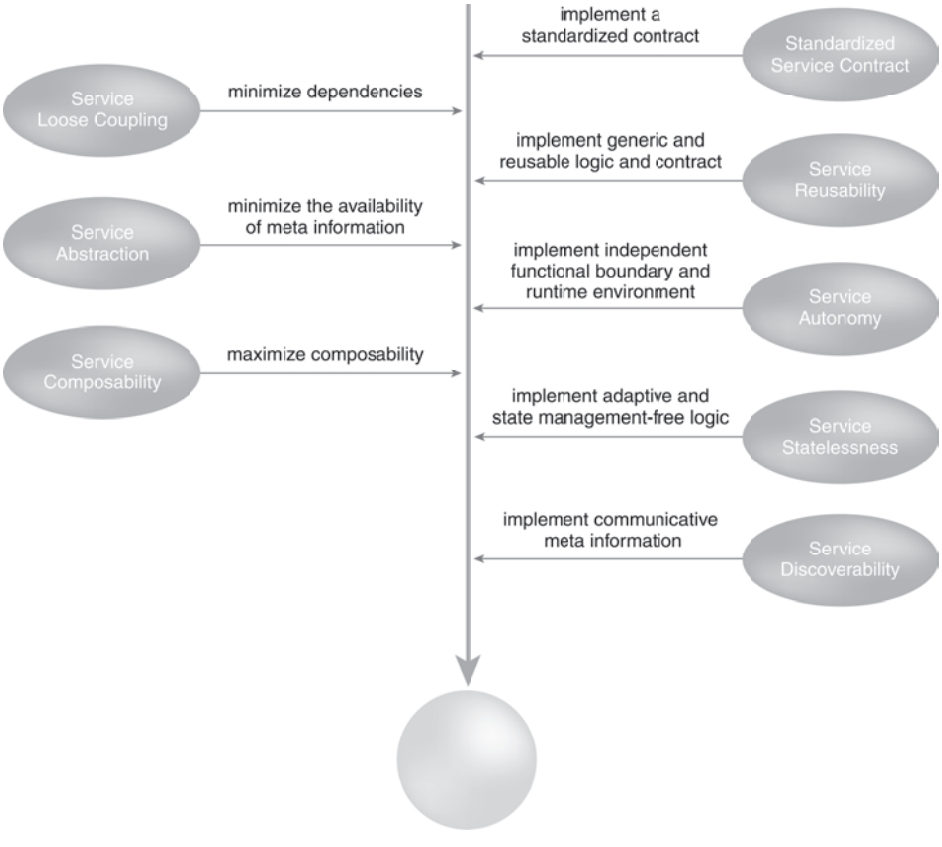



**Figure 3.9**
How service-orientation design principles can collectively shape service design.

Each of these design principles can, to some extent, influence how we decide to build a Web service contract. With regards to the topics covered in this book, the following principles have a direct impact.

### Standardized Service Contract

Given its name, it's quite evident that this design principle is only about service contracts and the requirement for them to be consistently standardized within the boundary of a service inventory. This design principle essentially advocates "contract first" design for services.

### Service Loose Coupling

This principle also relates to the service contract. Its design and how it is architecturally positioned within the service architecture are regulated with a strong emphasis on ensuring that only the right type of content makes its way into the contract in order to avoid the negative coupling types.

The following sections briefly describe common types of coupling. All are considered negative coupling types, except for the last.

#### *Contract-to-Functional Coupling*

Service contracts can become dependent on outside business processes, especially when they are coupled to logic that was designed directly in support of these processes. This can result in contract-to-functional coupling whereby the contract expresses characteristics that are specifically related to the parent process logic.

#### *Contract-to-Implementation Coupling*

When details about a service's underlying implementation are embedded within a service contract, an extent of contract-to-implementation coupling is formed. This negative coupling type commonly results when service contracts are a native part of the service implementation (as with component APIs) or when they are auto-generated and derived from implementation resources, such as legacy APIs, components, and databases.

#### *Contract-to-Logic Coupling*

The extent to which a service contract is bound to the underlying service programming logic is referred to as contract-to-logic coupling. This is considered a negative type of service coupling because service consumer programs that bind to the service contract end up also inadvertently forming dependencies on the underlying service logic.

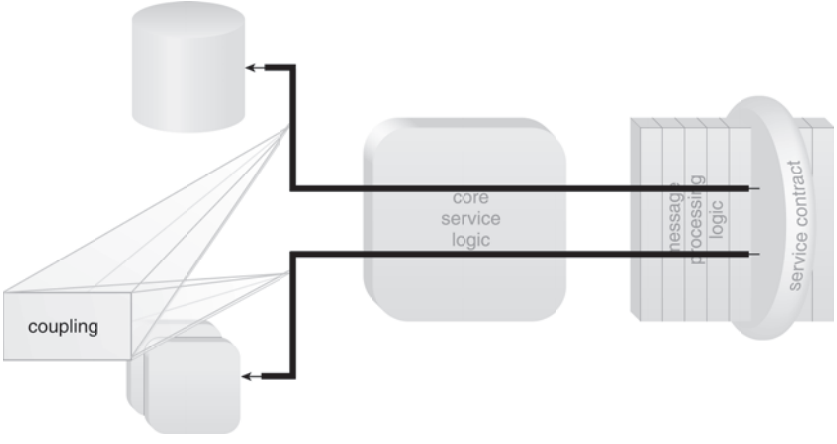



**Figure 3.10**
A Web service contract can be negatively coupled to various parts of the underlying service implementation.

### *Contract-to-Technology Coupling*

When the contract exposed by a service is bound to non-industry-standard communications technology, it forms an extent of contract-to-technology coupling. Although this coupling type could be applied to the dependencies associated with any proprietary technology, it is used exclusively for communications technology because that is what service contracts are generally concerned with.

An example of contract-to-technology coupling is when the service exists as a distributed component that requires the use of a proprietary RPC technology. Because this book is focused solely on Web service contract technology, this coupling type does not pose a design concern.

### *Logic-to-Contract Coupling*

Each of the previously described forms of coupling are considered negative because they can shorten the lifespan of a Web service contract, thereby leading to increased governance burden as a result of having to manage service contract versions.

This book is focused on providing the skills necessary to achieve high levels of *logic-to-contract* coupling by ensuring that the Web service contract can be designed with complete independence from the underlying Web service implementation.

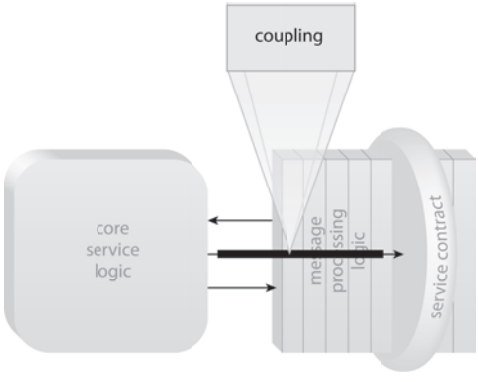



**Figure 3.11**
The most desirable design is for the Web service contract to remain an independent and fully decoupled part of the service architecture, thereby requiring the underlying logic to be coupled to it.

### Service Abstraction

By turning services into black boxes, the contracts are all that is officially made available to consumer designers who want to use the services. While much of this principle is about the controlled hiding of information by service owners, it also advocates the streamlining of contract content to ensure that only essential content is made available. The related use of the Validation Abstraction pattern further can affect aspects of contract design, especially related to the constraint granularity of service capabilities.

### Service Reusability

While this design principle is certainly focused on ensuring that service logic is designed to be robust and generic and much like a commercial product, these qualities also carry over into contract design. When viewing the service as a product and its contract as a generic API to which potentially many consumer programs will need to interface, the requirement emerges to ensure that the service's functional context, the definition of its capabilities, and the level at which each of its design granularities are set are appropriate for it to be positioned as a reusable enterprise resource.

### Service Discoverability

Because the service contracts usually represent all that is made available about a service, they are what this principle is primarily focused on when attempting to make each service as discoverable and interpretable as possible by a range of project team members.

Note that although Web service contracts need to be designed to be discoverable, this book does not discuss discovery processes or registry-based architectures.

#### Service Composability

This regulatory design principle is very concerned with ensuring that service contracts are designed to represent and enable services to be effective composition participants. The contracts must therefore adhere to the requirements of the previously listed design principles and also take multiple and complex service composition requirements into account.

#### Further Reading

Design principles are referenced throughout this book but represent a separate subject-matter that is covered in *SOA Principles of Service Design*. Introductory coverage of service-orientation as a whole is also available at SOAPrinciples.com.

### 3.4 SOA Design Patterns and Web Service Contracts

Design patterns provide proven solutions to common design problems. SOA has matured to an extent where a catalog of design patterns has been established. Of interest to us are those that affect the design and versioning of service contracts, specifically:

- *Canonical Expression* – Service contracts are standardized using naming conventions.
- *Canonical Schema* – Schema data models for common information sets are standardized across service contracts within a service inventory boundary.
- *Canonical Versioning* – Service contracts within the same service inventory are subject to the same versioning rules and conventions.
- *Compatible Change* – Already implemented service contracts are revised without breaking backwards compatibility.
- *Concurrent Contracts* – Multiple contracts can be created for a single service, each targeted at a specific type of consumer.
- *Contract Centralization* – Access to service logic is limited to the service contract, forcing consumers to avoid negative contract-to-implementation coupling.
- *Contract Denormalization* – Service contracts can include a measured extent of denormalization, allowing multiple capabilities to redundantly express core functions in different ways for different types of consumer programs.

- *Decomposed Capability* – Services prone to future decomposition can be equipped with a series of granular capabilities that more easily facilitate decomposition.
- *Decoupled Contract* – The service contract is physically decoupled from its implementation.
- *Distributed Capability* – The underlying service logic is distributed, thereby allowing the implementation logic for a capability with unique processing requirements to be physically separated, while continuing to be represented by the same service contract.
- *Messaging Metadata* – The message contents can be supplemented with activity-specific metadata that can be interpreted and processed separately at runtime.
- *Partial Validation* – Service consumers are designed to validate a subset of the data received from a service.
- *Policy Centralization* – Global or domain-specific policy assertions can be isolated and applied to multiple services.
- *Proxy Capability* – When a service contract needs to be decomposed, the original service contract can be preserved, even if underlying capability logic is separated, by turning the established capability definition into a proxy.
- *Schema Centralization* – Select schemas that exist as physically separate parts of the service contract are shared across multiple contracts.
- *Service Messaging* – Services can be designed to interact via a messaging-based technology, which removes the need for persistent connections and reduces coupling requirements.
- *Termination Notification* – Service contracts are extended to express termination information.
- *Validation Abstraction* – Granular validation logic and rules can be abstracted away from the service contract, thereby decreasing constraint granularity and increasing the contract's potential longevity.
- *Version Identification* – Version numbers and related information is expressed within service contracts.

*Web Services and the Decoupled Contract Pattern*

It is worth singling out Decoupled Contract at this stage because a Web service contract is essentially an implementation of this pattern. When building services as Web services, service contracts are positioned as physically separate parts of the service architecture. This allows us to fully leverage the technologies covered in this book in order to design and develop these contracts independently from the logic and implementations they will eventually represent.

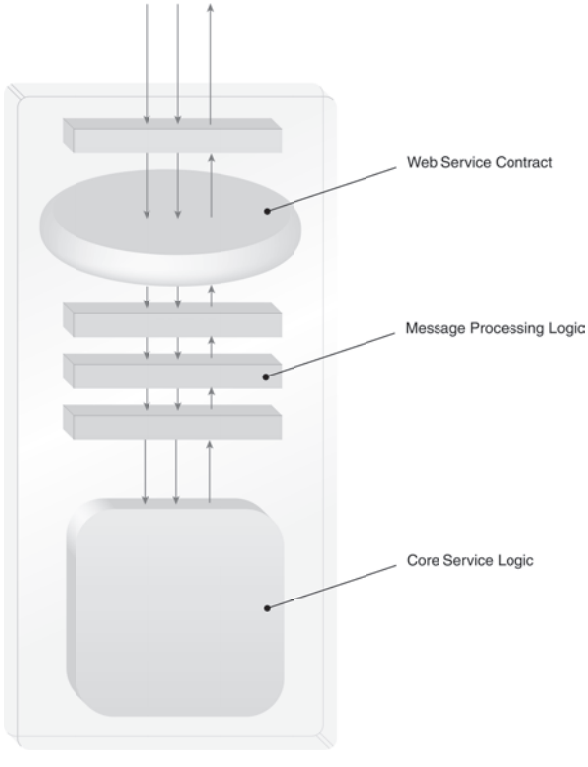



**Figure 3.12**
The Web service contract is a physically separated part of a Web service implementation.

Therefore, you may not see a lot of references to the Decoupled Contract pattern because it goes without saying that a Web service contract is naturally decoupled. However, it is always important to keep the coupling types explained earlier in the *Service-Orientation and Web Service Contracts* section in mind because although physically decoupled, the content of any Web service contract can still be negatively coupled to various parts of the service environment.

### Further Reading

The previously listed design patterns are part of a larger design pattern catalog published in the book *SOA Design Patterns*. Concise descriptions of these patterns are also available at SOAPatterns.org.

# The New Language of Business

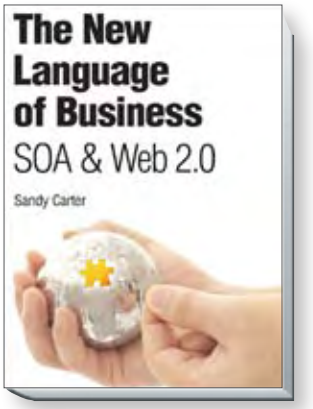
## SOA & Web 2.0

Sandy Carter

BUY ME

Sandy Carter

BUY ME




# The New Language of Business

SOA & Web 2.0

## Use ANGELS and Web 2.0 Marketing to Drive Powerful, Quantifiable Results

Today, marketers have an array of radically new Web 2.0-based techniques at their disposal. Now, leading IBM marketing innovator Sandy Carter introduces ANGELS, a start-tofinish framework for choosing the right Web 2.0 marketing tools—and using them to maximize revenue and profitability.

How you go from a whisper to a scream involves using market trends and new technologies to drive better results in a more cost effective way. Carter demonstrates winning Web 2.0 marketing at work through 54 brandnew case studies: organizations ranging fromStaples to Harley-Davidson, Coca-Cola to Mentos, Nortel to IBM itself.

> "Marketing has entered a new era of rapid advance. Those unwilling to experiment with new combinations of traditional and internet marketing will be left behind."
>
> — CHRIS TRIMBLE, Adjunct Associate Professor of Business Administration, Tuck School of Business at Dartmouth and Coauthor, *Ten Rules for Strategic Innovators: From Idea to Execution*

Includes information, case studies, and working examples for next generation marketing strategies such as:

- Social networks with virtual environments, including Second Life
- Online communities including Facebook
- Viral Marketing and eNurturing
- Serious Gaming
- Widgets
- Wikis
- Blogging, including Twitter
- RSS
- Podcasting
- Videocasting

AVAILABLE
- BOOK: 9780131956544
- SAFARI ONLINE Safari
- EBOOK: 0132353091
- KINDLE: B000SEGEVW

## 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. Sandy 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.

IBM Press™

ibmpressbooks.com

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.

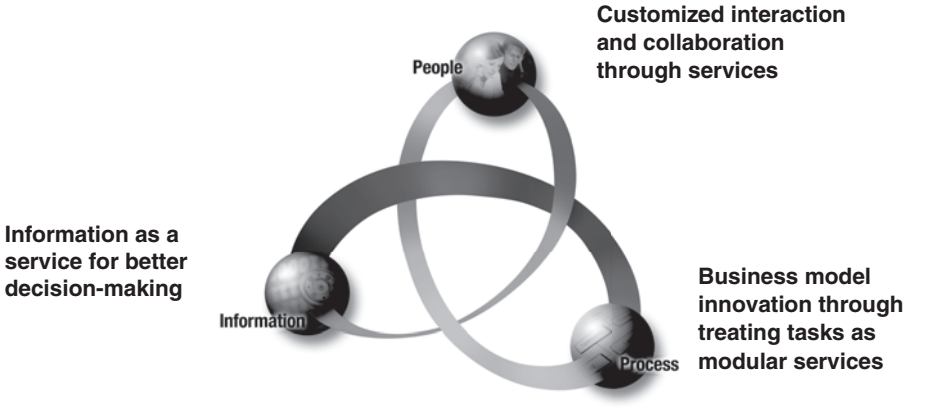



**Figure 11.1** Companies are taking an increasingly business-centric approach to SOA.

## 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.

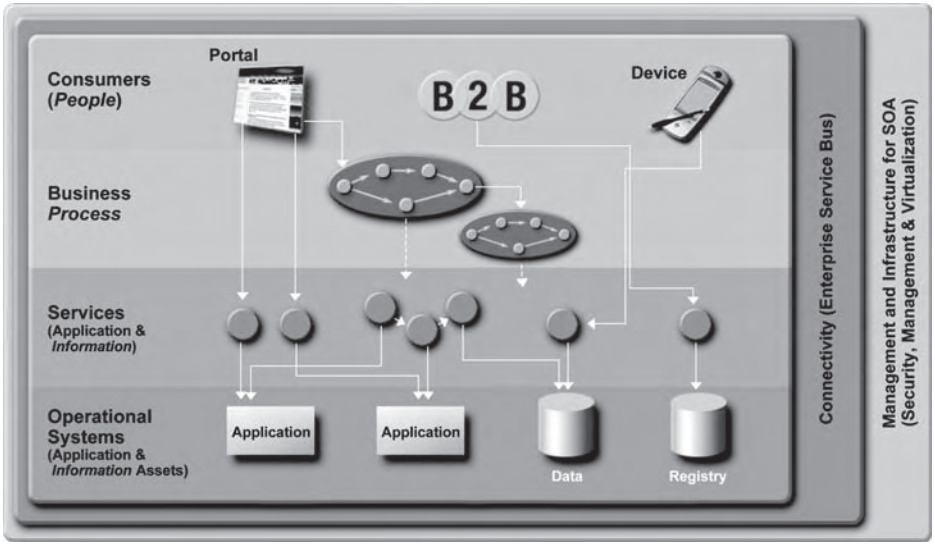
**The Benefits Are Much More Than the Sum of Their Parts**

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.

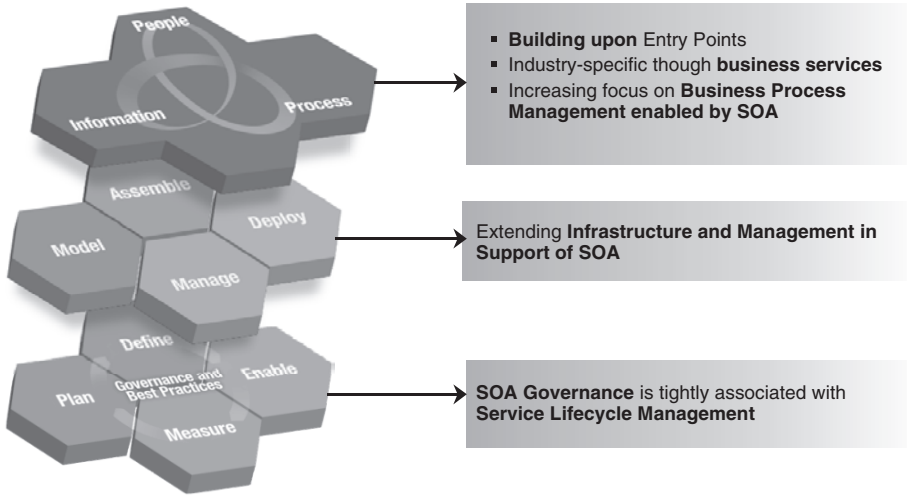



**Figure 11.3** The flex-pon-sive* agenda through SOA

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.

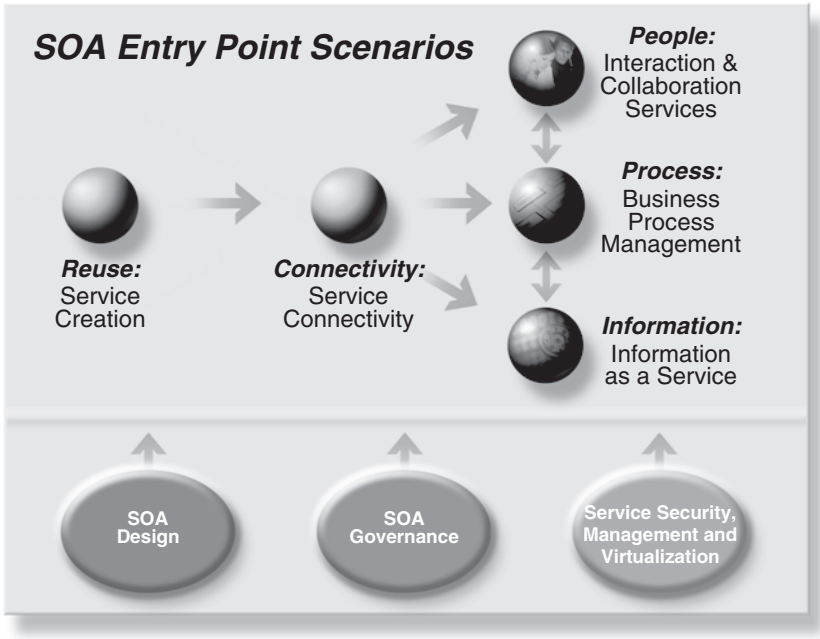



**Figure 11.4** SOA entry point scenarios based on more than 3,000 SOA engagements

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.

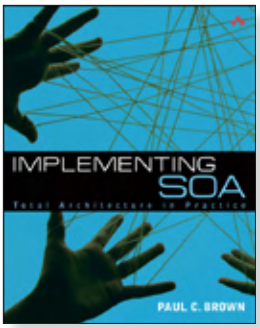
# MORE TITLES OF INTEREST


BUY ME

EVENT-DRIVEN ARCHITECTURE: How SOA Enables the Real-Time Enterprise

Hugh Taylor
Angela Yochem
Les Phillips
Frank Martinez

This book establishes the SOA-Event Driven Architecture (EDA) connection and lays out the specific ways in which SOA powers the EDA. While simple in context, the realities of bending SOAP Web Services into a functioning EDA are challenging and complex. The authors explain the theories and then put the reader on track for finding the right use of EDA in their organization and then getting it started.

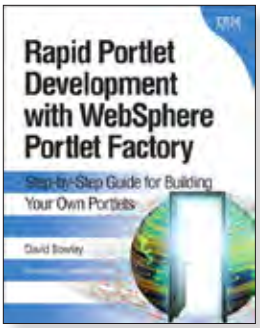



BUY ME

IMPLEMENTING SOA: Total Architecture in Practice

Paul C. Brown

Brown guides architects through the process of defining a service-oriented architecture at both the project and enterprise levels. He shows you how to smoothly integrate the design of both business processes and business systems, how to evolve existing architecture to achieve SOA objectives, and maintain operational support for the enterprise during the transition.




BUY ME

RAPID PORTLET DEVELOPMENT WITH WEBSPHERE PORTLET FACTORY: Step-by-Step Guide for Building Your Own Portlets

David Bowley

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."

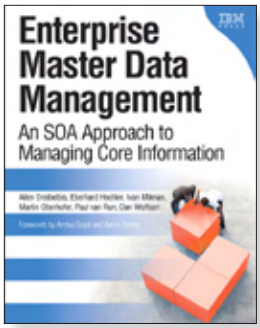

BUY ME

IBM WEBSPHERE DATAPOWER SOA APPLIANCE HANDBOOK

Bill Hines
John Rasmussen
Jaime Ryan
Simon Kapadia
Jim Brennan

Using many real-world examples, the authors systematically introduce the services available on DataPower devices, especially the "big three": XML Firewall, Web Service Proxy, and Multi-Protocol Gateway. They also present thorough and practical guidance on day-to-day DataPower management, including, monitoring, configuration build and deploy techniques.

# MORE TITLES OF INTEREST

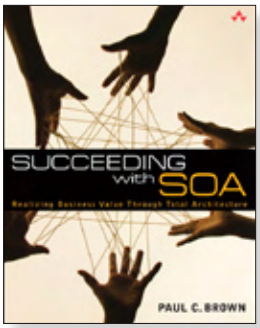



BUY ME

ENTERPRISE MASTER DATA MANAGEMENT:
An SOA Approach to Managing Core Information

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

This vendor-independent MDM technical reference systematically introduces MDM's key concepts and technical themes, explains its business case, and illuminates how it interrelates with and enables SOA. The authors introduce the MDM patterns, blueprints, solutions, and best practices you need to establish a consistent, manageable set of master data, and use it for competitive advantage.

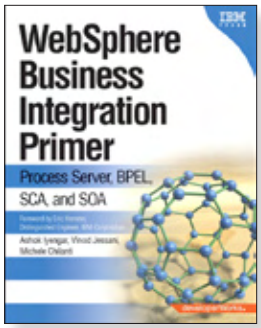



BUY ME

SUCCEEDING WITH SOA:
Realizing Business Value Through Total Architecture

Paul C. Brown

This book illustrates why total architecture is critical to enterprise success and the roles that the leadership team must play to make SOA work. *Succeeding with SOA* presents SOA from the management perspective and illustrates what can happen to enterprise business processes and projects when the elements of the total architecture are neglected.




BUY ME

WEBSPHERE BUSINESS INTEGRATION PRIMER:
Process Server, BPEL, SCA, and SOA

Ashok Iyengar
Vinod Jessani
Michele Chilanti

The authors thoroughly explain Service Component Architecture (SCA), basic business processes, and complex long-running business flows, and then introduce the key components of a WebSphere Business Integration (WBI) solution and show how to make them work together rapidly and efficiently.


BUY ME

APPLICATION ARCHITECTURE FOR WEBSPHERE:
A Practical Approach to Building WebSphere Applications

Joey Bernal

*Application Architecture 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.

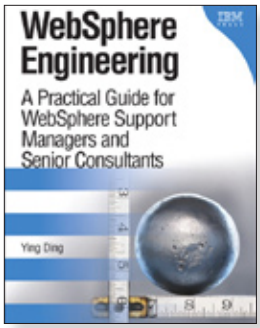
informIT.com
THE TRUSTED TECHNOLOGY LEARNING SOURCE

Addison Wesley

IBM Press

Visit **informit.com/learnsoa** for new releases, excerpts, and special offers.

# MORE TITLES OF INTEREST

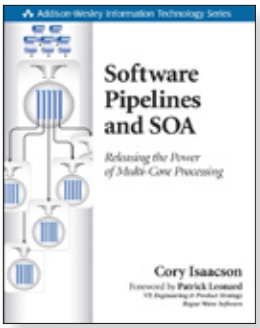



BUY ME

WEBSPHERE ENGINEERING:
A Practical Guide for WebSphere Support Managers and Senior Consultants

Ying Ding

Drawing on his tremendous real-world expertise, Ying Ding shows how to maximize the WebSphere platform's reliability, stability, scalability, and performance for large enterprise systems. You'll find insightful discussions of each option and strategy for managing WebSphere, including practical guidance on making the right tradeoffs for your environment.




BUY ME

SOFTWARE PIPELINES AND SOA:
Releasing the Power of Multi-Core Processing

Cory Isaacson

Through real-life business scenarios and examples of standard software pipelines design patterns, IT managers will learn how companies can scale applications to any size, maximize their resources, and most of all, maintain critical requirements that allow them to deliver on even their most challenging business objectives.


BUY ME

MASHUPS:
Strategies for the Modern Enterprise

J. Jeffrey Hanson

*Mashups* walks enterprise developers step-by-step through designing, coding, and debugging their first mashup. Hanson provides extensive code examples throughout, including a full chapter of case studies, as well as an insightful preview of the future of enterprise mashups.

PUBLISHING MAY 2009

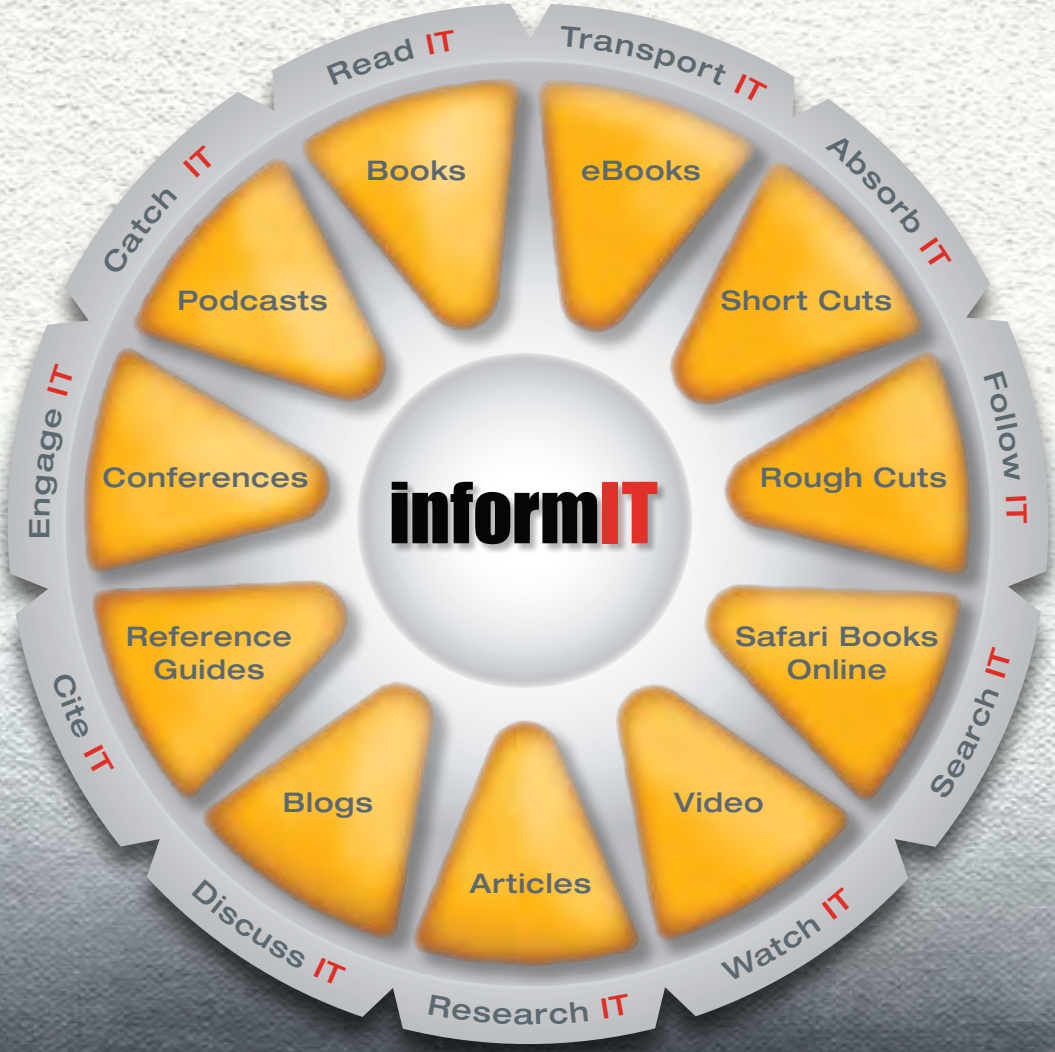
informIT.com
THE TRUSTED TECHNOLOGY LEARNING SOURCE

Addison Wesley

IBM Press

Visit **informit.com/learnsoa** for new releases, excerpts, and special offers.

# LearnIT at InformIT




## 11 WAYS TO LEARN IT AT www.InformIT.com/learn