Praise for This Book

"An absolute pleasure to read…the best SOA book I've read.

A book I would recommend to all of my colleagues; it provides much insight to the topics often overlooked by most books in this genre…the visuals were fantastic."

—Brandon Bohling, SOA Architecture and Strategy, Intel Corporation

"I recommend this book to any SOA practitioner who wishes to empower themselves in making service design real…gives readers the 360° view into service design [and] gives SOA practitioners the depth and understanding needed into the principles of SOA to assist in the design of a mature and successful SOA program."

—Stephen G. Bennett, Americas SOA Practice Lead, BEA Systems

"There are few references for SOA that give you the nuts and bolts and this one is at the top of the list. Well written and valuable as a reference book to any SOA practitioner."

—Dr. Mohamad Afshar, Director of Product Management,
Oracle Fusion Middleware, Oracle Corporation

"A very clear discussion of the subject matter. Provides a good structure that facilitates understanding and readily highlights key points."

—Kareem Yusuf, Director of SOA Strategy and Planning, IBM Software Group

"This book does a great job laying out benefits, key ideas and design principles behind successfully adopting service-oriented computing. At the same time, the book openly addresses challenges, risks and trade-offs that are in the way of adopting SOA in the real-world today. It moves away from ivory-tower views of service orientation, but still lays out a strong vision for SOA and outlines the changes necessary to realize the full potential."

—Christoph Schittko, Senior Architect, Microsoft

"This book strikes a healthy balance between theory and practice. It is a perfect complement to the SOA series by the author."

—Prakash Narayan, Sun Microsystems

"This book could be described as an encyclopedia of service design—Erl leaves nothing to chance. Indispensable."

—Steve Birkel, Chief IT Technical Architect, Intel Corp.

"I liked this book. It contains extremely important material for those who need to design services."

—Farzin Yashar, IBM SOA Advanced Technologies

"Thomas Erl's books are always densely filled with information that's well structured. This book is especially insightful for Enterprise Architects because it provide s great context and practical examples. Part 1 of the book alone is worth getting the book for."

—Markus Zirn, Senior Director, Product Management,
Oracle Fusion Middleware, Oracle Corporation

"This book is a milestone in SOA literature. For the first time we are provided with a practical guide on defining service characteristics and service design principles for SOA from a vendor-agnostic viewpoint. It's a great reference for SOA discovery, adoptions, and implementation projects."

—Canyang Kevin Liu, Principal Enterprise Architect, SAP Americas, Inc.

"There are very few who understand SOA like Thomas Erl does! The principle centric description of service orientation from Thomas canonizes the underpinnings of this important paradigm shift in creating agile and reusable software capabilities. The principles, so eloquently explained, leave little room for any ambiguity attached to the greater purpose of SOA. Most organizations today are creating services in a bottoms-up approach, realizing composition and reuse organically. The time is ripe for a book like this that prepares architects for a principle centric approach to SOA."

—Hanu Kommalapati, Architect, Microsoft Corporation

"If you are going to be designing, developing, or implementing SOA, this is a must have book."

—Jason "AJ" Comfort Sr., Booz Allen Hamilton

"An excellent book for anyone who wants to understand service-orientation and the principles involved in designing services…a clear, concise and articulate exploration of the eight design principles involved in analyzing, designing, implementing, and maintaining services…"

—Anish Karmarkar, Oracle Corporation

"Very well written, succinct, and easy to understand."

—Raj Balasubramanian, IBM Software Group

"A thorough examination of the considerations of service design. Both seasoned SOA practitioners and those endeavoring to realize services can benefit from reading this book."

—Bill Draven, Enterprise Architect, Intel Corporation

"I am very impressed. Comprehensive. Educative. This book helped me to step back and look at the SOA principles from broader perspective. I'd say this is a must-read book for SOA stakeholders."

—Radovan Janecek, Director R&D, SOA Center, Hewlett-Packard

"A comprehensive exploration of the issues of service design which has the potential to become the definitive work in this area."

—James Pasley, Chief Technology Officer, Cape Clear Software

"SOA projects are most successful when they are based on a solid technical foundation. Well accepted and established design principles are part of this foundation. This book takes a very structured approach at defining the core design principles for SOA, thus allowing the reader to immediately applying them to a project. Each principle is formally introduced and explained, and examples are given for how to apply it to a real design problem. A 'must read' for any architect, designer or developer of service oriented solutions".

—Andre Tost, Senior Technical Staff Member, IBM Software Group

"Outstanding SOA literature uniquely focused on the fundamental services design with thorough and in-depth study on all practical aspects from design principles to methodologies. This book provides a systematic approach for SOA adoption essential for both IT management and professionals."

—Robin Chen, PhD, Google, Inc.

"An excellent addition to any SOA library; it covers a wide range of issues in enough detail to be a valuable asset to anyone considering designing or using SOA based technologies."

—Mark Little, Director of Standards, Red Hat

"Very valuable guidance for understanding and applying SOA service design principles with concrete examples. A must read for the practitioner of SOA service design."

—Umit Yalcinalp, PhD, Standards Architect, SAP

"This book communicates complex concepts in a clear and concise manner. Examples and illustrations are used very effectively."

—Darryl Hogan, Senior Architect, Microsoft

"This book really does an excellent job of explaining the principles underpinning the value of SOA…Erl goes to great length to explain and give examples of each of the 8 principles that will significantly increase the readers ability to drive an SOA service design that benefits both business and IT."

—Robert Laird, IT Architect, IBM EAI/SOA Advanced Technologies Group

"A work of genius…Offers the most comprehensive and thorough explanation on the principles of service design and what it means to be 'service oriented.'

"Erl's treatment of the complex world of service oriented architecture is pragmatic, inclusive of real world situations and offers readers ways to communicate these ideas through illustrations and well formulated processes."

—David Michalowicz, MITRE Corporation

"This is the book for the large organization trying to rationalize its IT assets and establish an agile platform for the future. By highlighting risk and rewards, Thomas Erl brings clarity to how Service Orientation can be applied to ensure a responsive IT organization. This book finally brings software engineering principles to address the real world development challenges being faced.

To effectively serve the business, let alone embrace SOA, everyone involved should be familiar with the concepts investigated here. Thomas Erl thoroughly clarifies the nuances and defines the practice of service design.

We expect that this will become a classic text in software engineering, corporate training and colleges."

—Cory Isaacson, President, Rogue Wave Software and Ravi Palepu, SOA Author and Speaker

"Thomas Erl does a great job…an easy read."

—Michael H. Sor, Booz Allen Hamilton

"…a must read for SOA Architects to develop a firm foundation and understanding of the principles (and trade-offs) that make up a good SOA service.

After reading this book, it finally 'clicked' as to why a properly designed SOA system is different (and better) than a system based on previous enterprise architectures."

—Fred Ingham, Platinum Solutions Inc.

"Lays a tremendous foundation for business and technical workers to come to common terms and expectations…incredibly enlightening to see the details associated with achieving the SOA vision."

—Wayne P. Ariola, Vice President of Strategy, Parasoft

"[Erl does] and excellent job of addressing the breadth of [his] audience to present to those new to SOA and weaved in enough detail to assist those who are already actively involved in SOA development."

—R. Perry Smith, Application Program Manager, EDS/OnStar

"It is easy to miss the big picture of what SOA means for the design of larger scale systems amidst the details of WS technologies. Erl helps provide a broader perspective, surveying the landscape from a design standpoint."

—Jim Clune, Chief Architect, Parasoft

"Lays a firm foundation for the underlying principles of good service design. Cuts through the hype and provides a cogent resource for improving architectural judgment on SOA projects."

—Jim Murphy, Vice President of Product Management, Mindreef, Inc.

"The first book to concisely, gradually and comprehensively explain how to apply SOA principles into enterprise-level software design. It is an excellent book."

—Robin G. Qiu, Ph.D., Division of Engineering and Information Science, Pennsylvania State University

"I really think that this is a very useful book that a lot of people really need out there in the industry."

—Dr. Arnaud Simon, Principal Software Engineer, Red Hat

"…indispensable companion to designing and implementing a service-oriented architecture. It condenses all information necessary to design services and is the most relevant source I know if in the field."

"[This book is] not only helpful, but fundamental to successfully designing an SOA."

—Phillipp Offermann, Research Analyst, University of Berlin

"Service-Oriented Architecture is an important topic in IT today. Its vast scope could span an organization's enterprise. Designing it properly is a major undertaking. This book provides timely, expert and comprehensive discussions on the principles of service design. Thomas has a keen sense in identifying the subtle points of various subjects and explains them in an easy-to-understand way. The book is a valuable resource for IT professionals working in SOA."

—Peter H. Chang, PhD, Associate Professor of Information Systems, Lawrence Technological University

SOA: Principles of Service Design

The Service-Oriented Computing Series from Thomas Erl aims to provide the IT industry with a consistent level of unbiased, practical, and comprehensive guidance and instruction in the areas of service-oriented architecture, service-orientation, and the expanding landscape that is shaping the real-world service-oriented computing platform.

For more information, visit www.soabooks.com.

SOA
Principles of Service Design

Thomas Erl

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

Editor-in-Chief
Mark L. Taub

Managing Editor
Gina Kanouse

Senior Project Editor
Kristy Hart

Copy Editor
Language Logistics, LLC

Senior Indexer
Cheryl Lenser

Proofreader
Williams Woods Publishing

Publishing Coordinator
Noreen Regina

Compositor
Jake McFarland

Cover Designer
Thomas Erl

Graphics
Zuzana Cappova
Spencer Fruhling

Photos
Thomas Erl

Library of Congress Cataloging-in-Publication Data

Erl, Thomas.
SOA: principles of service design / Thomas Erl.
p. cm.
ISBN 0-13-234482-3 (hardback : alk. paper) 1. Web services. 2. Computer architecture. 3. System analysis. 4. System design. I. Title.
TK5105.88813.E75 2008
004.2'2—dc22

ISBN-13: 9780132344821
ISBN-10: 0132344823
Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
Ninth Printing: May 2014

To my wife and family for their support.

This page intentionally left blank

Contents

PART II: DESIGN PRINCIPLES

PART III: SUPPLEMENTAL

Preface

Over the past few years I've been exposed to many different IT environments as part of a wide range of SOA initiatives for clients in both private and public sectors. While doing some work on a project for a client in the defense industry, I had an opportunity to learn more about not just their technical landscape, but also the various policies and procedures that are specific to the defense culture. During this time I came across the DoD Standardization Program, an initiative comprised of documents and specifications that establish guiding principles and standards for various aspects of the military, including the design of weapons and military equipment, as well as the definition of methods and processes used by military personnel.

While reading about this program, I learned that several other standardization programs have been in existence for some time, facilitating standardization within public sector organizations (such as the Coast Guard and NASA), as well as numerous private sector industries. The goals of these programs tend to revolve around the establishment of industry standards to enhance interoperability with the ultimate objective of reducing operational overhead, reducing risk, and increasing the organization's overall effectiveness.

In the case of the aforementioned public sector-related standards, interoperability may refer to the exchange of equipment or weapons or the exchange and collaboration of personnel from different locations.

For example, an ammunition clip manufactured in Iowa, stored in Virginia, and delivered to and used by someone at a training base in Texas will work perfectly with a gun manufactured in Kansas because both of these products were built according to the same set of specifications. Similarly, in response to a natural disaster a rescue team may need to be quickly assembled from individuals based out of different cities and who have never previously worked together. This team can still function effectively because all team members were trained as per the same procedures and processes, using the same vocabulary and conventions.

These standardization programs have much in common with the rationale and objectives behind SOA and service-orientation. The fundamental goal is to produce something with repeatable value, long-term benefit, and inherent flexibility, all for the strategic good of the organization. The greatest obstacle to achieving this goal in the world of SOA has been a lack of understanding as to what service-orientation, as an industry paradigm, really is. It is my hope that this book will help rectify this situation by providing some clarity for what it means for something to be “service-oriented.”

Acknowledgments

To ensure the accuracy and legitimacy of the content in this book, I decided early on to subject it to a rigorous quality assurance process that involved technical reviews by over 60 industry professionals. I am deeply grateful for the time and effort these individuals dedicated to these reviews. Specifically, I would like to thank Kevin Davis, PhD, Ronald Bourret, Robert Schneider, Ravi Palepu, Wes McGregor, Judith Myerson, and Cyrille Thilloy for their early feedback, and the following technical reviewers that participated in the full manuscript review (in alphabetical order by last name):

Dr. Mohamad Afshar, Oracle Corporation

Wayne Ariola, Parasoft

Raj Balasubramanian, IBM Software Group

Stephen Bennett, BEA Systems, Inc.

Steve Birkel, Intel Corporation

Brandon Bohling, Intel Corporation

Peter Chang, PhD, Lawrence Technological University

Robin Chen, PhD, Google, Inc.

Jim Clune, Parasoft

Jason "AJ" Comfort Sr., Booz Allen Hamilton, Inc.

Bill Draven, Intel Corporation

Darryl Hogan, Microsoft Corporation

Continues

Fred Ingham, Platinum Solutions Inc.

Cory Isaacson, Rogue Wave Software

Radovan Janecek, Hewlett-Packard

Anish Karmarkar, Oracle Corporation

Hanu Kommalapati, Microsoft Corporation

Robert Laird, IBM EAI/SOA Advanced Technologies Group

Dr. Mark Little, Redhat

Canyang Kevin Liu, SAP Americas, Inc.

David Michalowicz, MITRE Corporation

Jim Murphy, Mindreef, Inc.

Prakash Narayan, Sun Microsystems

Philipp Offermann, University of Berlin

James Pasley, Cape Clear Software

Robin G. Qiu, PhD, Pennsylvania State University

Christoph Schittko, Microsoft Corporation

Dr. Arnaud Simon, Redhat

R. Perry Smith, EDS/OnStar

Michael H. Sor, Booz Allen Hamilton, Inc.

Philip Thomas, IBM United Kingdom Limited

Andre Tost, IBM Software Group

Sameer Tyagi, Fidelity Investments

Umit Yalcinalp, SAP

Farzin Yashar, IBM SOA Advanced Technologies

Kareem Yusuf, IBM Software Group

Markus Zirn, Oracle Corporation

Chapter 4

Service-Orientation

Having covered some of the basic elements of service-oriented computing, we now narrow our focus on service-orientation. The next set of sections establish the paradigm of service-orientation and explain how it is changing the face of distributed computing.

4.1 Introduction to Service-Orientation

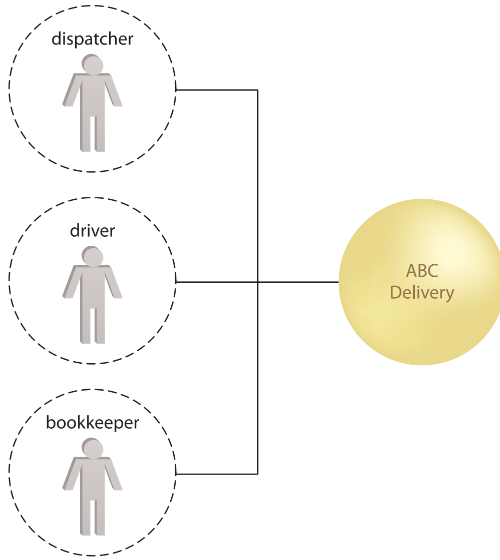
In the every day world around us, services are and have been commonplace for as long as civilized history has existed. Any person carrying out a distinct task in support of others is providing a service (Figure 4.1). Any group of individuals collectively performing a task is also demonstrating the delivery of a service.

Figure 4.1
Three individuals, each capable of providing a distinct service.

Similarly, an organization that carries out tasks associated with its purpose or business is also providing a service. As long as the task or function being provided is well-defined and can be relatively isolated from other associated tasks, it can be distinctly classified as a service (Figure 4.2).

Certain baseline requirements exist to enable a group of individual service providers to collaborate in order to collectively provide a larger service. Figure 4.2, for example, displays a group of employees that each provide a service for ABC Delivery. Even though each individual contributes a distinct service, for the company to function effectively, its staff also needs to have fundamental, common characteristics, such as availability, reliability, and the ability to communicate using the same language. With all of this in place, these individuals can be composed into a productive working team. Establishing these types of baseline requirements is a key goal of service-orientation.

Figure 4.2
A company that employs these three people can compose their capabilities to carry out its business.



Services in Business Automation

In the world of SOA and service-orientation, the term "service" is not generic. It has specific connotations that relate to a unique combination of design characteristics. When solution logic is consistently built as services and when services are consistently designed with these common characteristics, service-orientation is successfully realized throughout an environment.

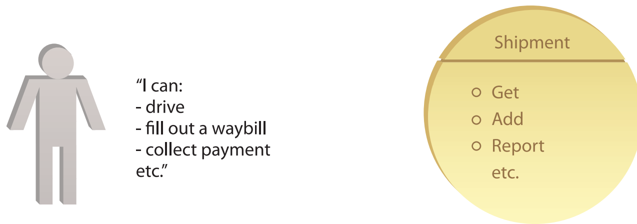
For example, one of the primary service design characteristics explored as part of this study of service-orientation is reusability. A strong emphasis on producing solution logic in the format of services that are positioned as highly generic and reusable enterprise resources gradually transitions an organization to a state where more and more of its solution logic becomes less dependent on and more agnostic to any one purpose or business process. Repeatedly fostering this characteristic within services eventually results in wide-spread reuse potential.

Consistently realizing specific design characteristics requires a set of guiding principles. This is what the service-orientation design paradigm is all about.

Services Are Collections of Capabilities

When discussing services, it is important to remember that a single service can provide a collection of capabilities. They are grouped together because they relate to a functional context established by the service. The functional context of the service illustrated in Figure 4.3, for example, is that of "shipment." Therefore, this particular service provides a set of capabilities associated with the processing of shipments.

Figure 4.3
Much like a human, an automated service can provide multiple capabilities.



A service can essentially act as a container of related capabilities. It is comprised of a body of logic designed to carry out these capabilities and a service contract that expresses which of its capabilities are made available for public invocation.

References to service capabilities in this book are specifically focused on those that are defined in the service contract. For a discussion of how service capabilities are distinguished from Web service operations and component methods, see the *Principles and Service Implementation Mediums* section in Chapter 5.

Service-Orientation as a Design Paradigm

As established in Chapter 3, a design paradigm is an approach to designing solution logic. When building distributed solution logic, design approaches revolve around a software engineering theory known as the *separation of concerns*. In a nutshell, this theory states that a larger problem is more effectively solved when decomposed into a set of smaller problems or *concerns*. This gives us the option of partitioning solution logic into capabilities, each designed to solve an individual concern. Related capabilities can be grouped into units of solution logic.

The fundamental benefit to solving problems this way is that a number of the solution logic units can be designed to solve immediate concerns while still remaining agnostic to the greater problem. This provides the constant opportunity for us to reutilize the capabilities within those units to solve other problems as well.

Different design paradigms exist for distributed solution logic. What distinguishes service-orientation is the manner in which it carries out the separation of concerns and how it shapes the individual units of solution logic. Applying service-orientation to a meaningful extent results in solution logic that can be safely classified as "service-oriented" and units that qualify as "services." To understand exactly what that means requires an appreciation of the strategic goals covered in Chapter 3 combined with knowledge of the associated design principles documented in Part II.

For now, let's briefly introduce each of these principles:

Standardized Service Contract

Services express their purpose and capabilities via a service contract. The Standardized Service Contract design principle is perhaps the most fundamental part of service-orientation in that it essentially requires that specific considerations be taken into account when designing a service's public technical interface and assessing the nature and quantity of content that will be published as part of a service's official contract.

A great deal of emphasis is placed on specific aspects of contract design, including the manner in which services express functionality, how data types and data models are defined, and how policies are asserted and attached. There is a constant focus on ensuring that service contracts are both optimized, appropriately granular, and standardized to ensure that the endpoints established by services are consistent, reliable, and governable.

Chapter 6 is dedicated to exploring this design principle in detail.

Service Loose Coupling

Coupling refers to a connection or relationship between two things. A measure of coupling is comparable to a level of dependency. This principle advocates the creation of a specific type of relationship within and outside of service boundaries, with a constant emphasis on reducing ("loosening") dependencies between the service contract, its implementation, and its service consumers.

The principle of Service Loose Coupling promotes the independent design and evolution of a service's logic and implementation while still guaranteeing baseline interoperability with consumers that have come to rely on the service's capabilities. There are numerous types of coupling involved in the design of a service, each of which can impact the content and granularity of its contract. Achieving the appropriate level of coupling requires that practical considerations be balanced against various service design preferences.

Chapter 7 provides an in-depth exploration of this principle and introduces related patterns and concepts.

Service Abstraction

Abstraction ties into many aspects of service-orientation. On a fundamental level, this principle emphasizes the need to hide as much of the underlying details of a service as possible. Doing so directly enables and preserves the previously described loosely coupled relationship. Service Abstraction also plays a significant role in the positioning and design of service compositions.

Various forms of meta data come into the picture when assessing appropriate abstraction levels. The extent of abstraction applied can affect service contract granularity and can further influence the ultimate cost and effort of governing the service.

Chapter 8 covers several aspects of applying abstraction to different types of service meta data, along with processes and approaches associated with information hiding.

Service Reusability

Reuse is strongly advocated within service-orientation; so much so, that it becomes a core part of typical service analysis and design processes, and also forms the basis for key service models. The advent of mature, non-proprietary service technology has provided the opportunity to maximize the reuse potential of multi-purpose logic on an unprecedented level.

The principle of Service Reusability emphasizes the positioning of services as enterprise resources with agnostic functional contexts. Numerous design considerations are raised to ensure that individual service capabilities are appropriately defined in relation to an agnostic service context, and to guarantee that they can facilitate the necessary reuse requirements.

Variations and levels of reuse and associated agnostic service models are covered in Chapter 9, along with a study of how commercial product design approaches have influenced this principle.

Service Autonomy

For services to carry out their capabilities consistently and reliably, their underlying solution logic needs to have a significant degree of control over its environment and resources. The principle of Service Autonomy supports the extent to which other design principles can be effectively realized in real world production environments by fostering design characteristics that increase a service’s reliability and behavioral predictability.

This principle raises various issues that pertain to the design of service logic as well as the service's actual implementation environment. Isolation levels and service normalization considerations are taken into account to achieve a suitable measure of autonomy, especially for reusable services that are frequently shared.

Chapter 10 documents the design issues and challenges related to attaining higher levels of service autonomy, and further classifies different forms of autonomy and highlights associated risks.

Service Statelessness

The management of excessive state information can compromise the availability of a service and undermine its scalability potential. Services are therefore ideally designed to remain stateful only when required. Applying the principle of Service Statelessness requires that measures of realistically attainable statelessness be assessed, based on the adequacy of the surrounding technology architecture to provide state management delegation and deferral options.

Chapter 11 explores the options and impacts of incorporating stateless design characteristics into service architectures.

Service Discoverability

For services to be positioned as IT assets with repeatable ROI they need to be easily identified and understood when opportunities for reuse present themselves. The service design therefore needs to take the "communications quality" of the service and its individual capabilities into account, regardless of whether a discovery mechanism (such as a service registry) is an immediate part of the environment.

The application of this principle, as well as an explanation of how discoverability relates to interpretability and the overall service discovery process, are covered in Chapter 12.

Service Composability

As the sophistication of service-oriented solutions continues to grow, so does the complexity of underlying service composition configurations. The ability to effectively compose services is a critical requirement for achieving some of the most fundamental goals of service-oriented computing.

Complex service compositions place demands on service design that need to be anticipated to avoid massive retro-fitting efforts. Services are expected to be capable of participating as effective composition members, regardless of whether they need to be immediately enlisted in a composition. The principle of Service Composability addresses this requirement by ensuring that a variety of considerations are taken into account.

How the application of this design principle helps prepare services for the world of complex compositions is described in Chapter 13.

Service-Orientation and Interoperability

One item that may appear to be absent from the preceding list is a principle along the lines of *"Services are Interoperable."* The reason this does not exist as a separate principle is because interoperability is fundamental to every one of the principles just described. Therefore, in relation to service-oriented computing, stating that services must be interoperable is just about as basic as stating that services must exist. Each of the eight principles supports or contributes to interoperability in some manner.

Here are just a few examples:

- Service contracts are standardized to guarantee a baseline measure of interoperability associated with the harmonization of data models.
- Reducing the degree of service coupling fosters interoperability by making individual services less dependent on others and therefore more open for invocation by different service consumers.
- Abstracting details about the service limits all interoperation to the service contract, increasing the long-term consistency of interoperability by allowing underlying service logic to evolve more independently.
- Designing services for reuse implies a high-level of required interoperability between the service and numerous potential service consumers.
- By raising a service's individual autonomy, its behavior becomes more consistently predictable, increasing its reuse potential and thereby its attainable level of interoperability.
- Through an emphasis on stateless design, the availability and scalability of services increase, allowing them to interoperate more frequently and reliably.

- Service Discoverability simply allows services to be more easily located by those who want to potentially interoperate with them.
- Finally, for services to be effectively composable they must be interoperable. The success of fulfilling composability requirements is often tied directly to the extent to which services are standardized and cross-service data exchange is optimized.

A fundamental goal of applying service-orientation is for interoperability to become a natural by-product, ideally to the extent that a level of intrinsic interoperability is established as a common and expected service design characteristic. Depending on the architectural strategy being employed, this extent may or may not be limited to a specific service inventory.

Of course, as with any other design characteristic, there are levels of interoperability a service can attain. The ultimate measure is generally determined by the extent to which service-orientation principles have been consistently and successfully realized (plus, of course, environmental factors such as the compatibility of wire protocols, the maturity level of the underlying technology platform, and adherence to technology standards).

NOTE

Increased intrinsic interoperability is one of the key strategic goals associated with service-oriented computing (as originally established in Chapter 3). For more detailed information about how service-orientation principles directly support this and other strategic goals, see Chapter 16.

SUMMARY OF KEY POINTS

- The service-orientation paradigm consists of eight distinct design principles, each of which fosters fundamental design characteristics, such as interoperability. These principles are explored individually in subsequent chapters.
- Interoperability is a natural by-product of applying service-orientation design principles.

4.2 Problems Solved by Service-Orientation

To best appreciate why service-orientation has emerged and how it is intended to improve the design of automation systems, we need to compare before and after perspectives. By studying some of the common issues that have historically plagued IT, we can begin to understand the solutions proposed by this design paradigm.

> **NOTE**
>
> This book fully acknowledges that past design paradigms have advocated similar principles and strategic goals as service-orientation. Several of these design approaches, in fact, directly inspired or influenced service-orientation (as explained further in the *Origins and Influences of Service-Orientation* section of this chapter). The following section is focused specifically on a comparison with the silo-based design approach because it has persisted as the most common means by which applications are delivered.

Life Before Service-Orientation

In the world of business it makes a great deal of sense to deliver solutions capable of automating the execution of business tasks. Over the course of IT's history, the majority of such solutions have been created with a common approach of identifying the business tasks to be automated, defining their business requirements, and then building the corresponding solution logic (Figure 4.4).

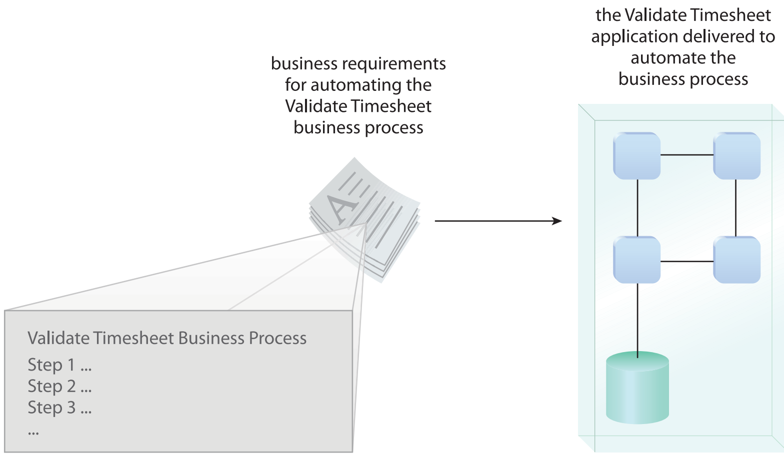


Figure 4.4
A ratio of one application for each new set of automation requirements has been common.

This has been an accepted and proven approach to achieving tangible business benefits through the use of technology and has been successful at providing a relatively predictable return on investment (Figure 4.5).

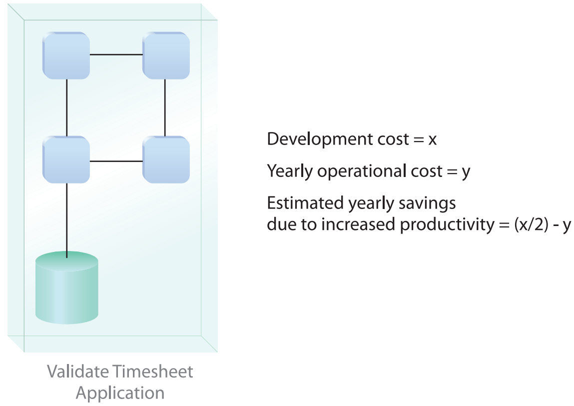


Figure 4.5
A sample formula for calculating ROI is based on a predetermined investment with a predictable return.

The ability to gain any further value from these applications is usually inhibited because their capabilities are tied to specific business requirements and processes (some of which will even have a limited lifespan). When new requirements and processes come our way, we are forced to either make significant changes to what we already have, or we may need to build a new application altogether.

In the latter case, although repeatedly building "disposable applications" is not the perfect approach, it has proven itself as a legitimate means of automating business. Let's explore some of the lessons learned by first focusing on the positive.

- Solutions can be built efficiently because they only need to be concerned with the fulfillment of a narrow set of requirements associated with a limited set of business processes.
- The business analysis effort involved with defining the process to be automated is straight forward. Analysts are focused only on one process at a time and therefore only concern themselves with the business entities and domains associated with that one process.
- Solution designs are tactically focused. Although complex and sophisticated automation solutions are sometimes required, the sole purpose of each is to automate just one or a specific set of business processes. This predefined functional scope simplifies the overall solution design as well as the underlying application architecture.

- The project delivery lifecycle for each solution is streamlined and relatively predictable. Although IT projects are notorious for being complex endeavors, riddled with unforeseen challenges, when the delivery scope is well-defined (and doesn't change), the process and execution of the delivery phases have a good chance of being carried out as expected.
- Building new systems from the ground up allows organizations to take advantage of the latest technology advancements. The IT marketplace progresses every year to the extent that we fully expect technology we use to build solution logic today to be different and better tomorrow. As a result, organizations that repeatedly build disposable applications can leverage the latest technology innovations with each new project.

These and other common characteristics of traditional solution delivery provide a good indication as to why this approach has been so popular. Despite its acceptance, though, it has become evident that there is still lots of room for improvement.

It Can Be Highly Wasteful

The creation of new solution logic in a given enterprise commonly results in a significant amount of redundant functionality (Figure 4.6). The effort and expense required to construct this logic is therefore also redundant.

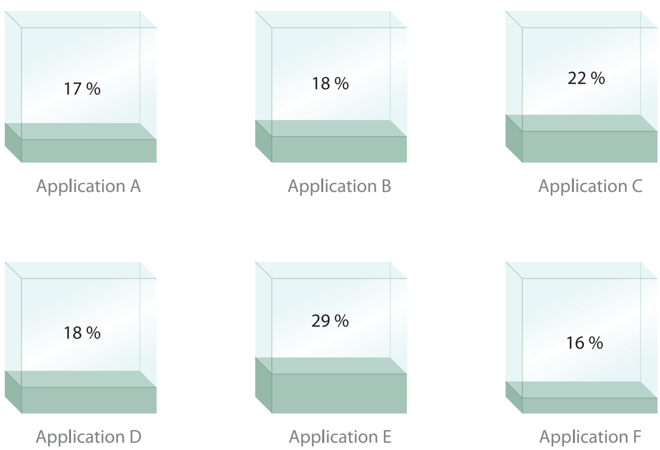


Figure 4.6

Different applications developed independently can result in significant amounts of redundant functionality. The applications displayed were delivered with various levels of solution logic that, in some form, already existed.

It's Not as Efficient as it Appears

Because of the tactical focus on delivering solutions for specific process requirements, the scope of development projects is highly targeted. Therefore, there is the constant perception that business requirements will be fulfilled at the earliest possible time. However, by continually building and rebuilding logic that already exists elsewhere, the process is not as efficient as it could be if the creation of redundant logic could be avoided (Figure 4.7).

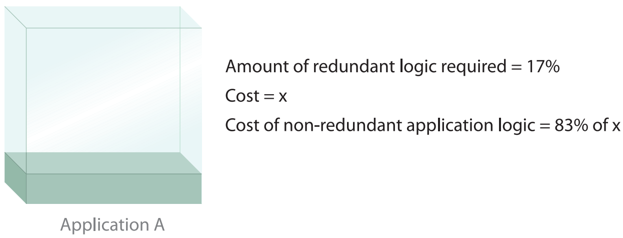


Figure 4.7
Application A was delivered for a specific set of business requirements. Because a subset of these business requirements had already been fulfilled elsewhere, Application A's delivery scope is larger than it has to be.

It Bloats an Enterprise

Each new or extended application adds to the bulk of an IT environment's system inventory (Figure 4.8). The ever-expanding hosting, maintenance, and administration demands can inflate an IT department in budget, resources, and size to the extent that IT becomes a significant drain on the overall organization.

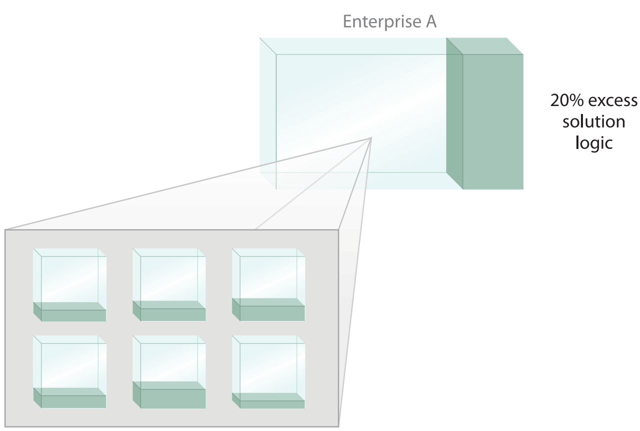


Figure 4.8
This simple diagram portrays an enterprise environment containing applications with redundant functionality. The net effect is a larger enterprise.

It Can Result in Complex Infrastructures and Convoluted Enterprise Architectures

Having to host numerous applications built from different generations of technologies and perhaps even different technology platforms often requires that each will impose unique architectural requirements. The disparity across these “siloed” applications can lead to a counter-federated environment (Figure 4.9), making it challenging to plan the evolution of an enterprise and scale its infrastructure in response to that evolution.

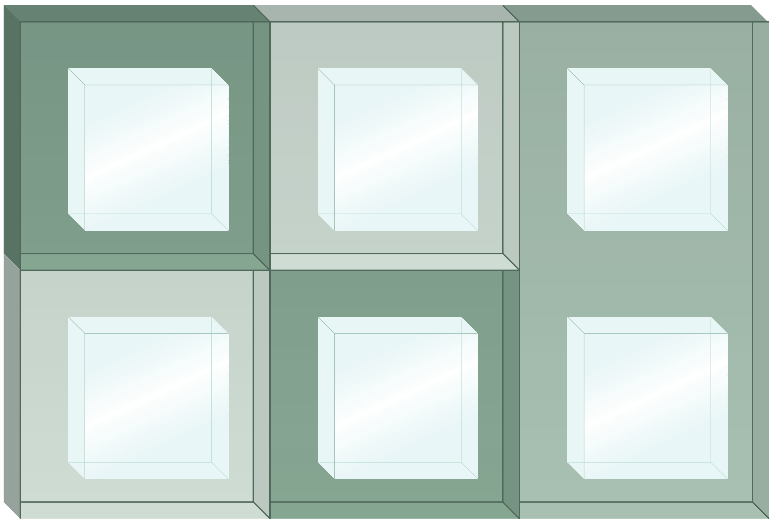


Figure 4.9
Different application environments within the same enterprise can introduce incompatible runtime platforms as indicated by the shaded zones.

Integration Becomes a Constant Challenge

Applications built only with the automation of specific business processes in mind are generally not designed to accommodate other interoperability requirements. Making these types of applications share data at some later point results in a jungle of convoluted integration architectures held together mostly through point-to-point patchwork (Figure 4.10) or requiring the introduction of large middleware layers.

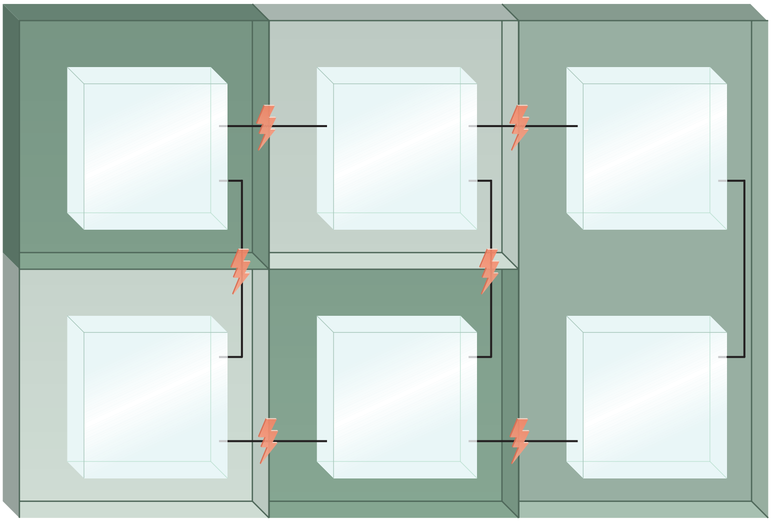


Figure 4.10
A vendor-diverse enterprise can introduce a variety of integration challenges, as expressed by the little lightning bolts that highlight points of concern when trying to bridge proprietary environments.

The Need for Service-Orientation

After repeated generations of traditional distributed solutions, the severity of the previously described problems has been amplified. This is why service-orientation was conceived. It very much represents an evolutionary state in the history of IT in that it combines successful design elements of past approaches with new design elements that leverage conceptual and technology innovation.

The consistent application of the eight design principles listed earlier results in the widespread proliferation of the corresponding design characteristics:

- increased consistency in how functionality and data is represented
- reduced dependencies between units of solution logic
- reduced awareness of underlying solution logic design and implementation details
- increased opportunities to use a piece of solution logic for multiple purposes
- increased opportunities to combine units of solution logic into different configurations

- increased behavioral predictability
- increased availability and scalability
- increased awareness of available solution logic

When these characteristics exist as real parts of implemented services, they establish a common synergy. As a result, the complexion of an enterprise changes as the following distinct qualities are consistently promoted:

Increased Amounts of Agnostic Solution Logic

Within a service-oriented solution, units of logic (services) encapsulate functionality not specific to any one application or business process (Figure 4.11). These services are therefore classified as agnostic and reusable IT assets.

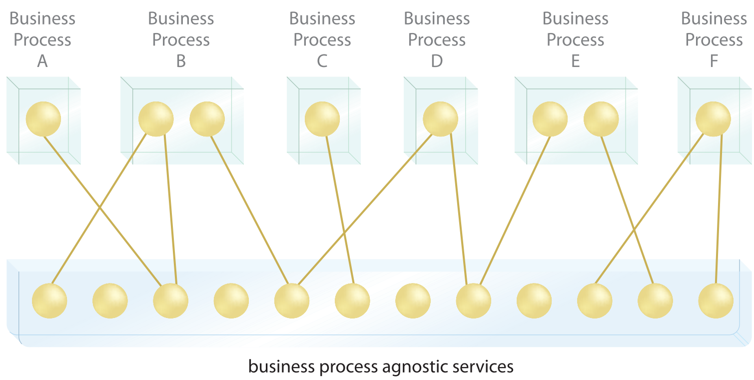


Figure 4.11
Business processes are automated by a series of business process-specific services (top layer) that share a pool of business process-agnostic services (bottom layer). These layers correspond to the task, entity, and utility service models described in Chapter 3.

Reduced Amounts of Application-Specific Logic

Increasing the amount of solution logic not specific to any one application or business process decreases the amount of required application-specific logic (Figure 4.12). This blurs the lines between standalone application environments by reducing the overall quantity of standalone applications. (See also the *Service-Orientation and the Concept of "Application"* section later in this chapter.)

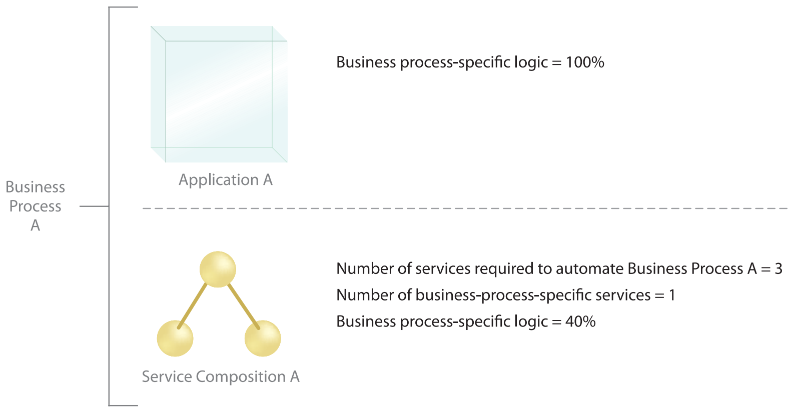


Figure 4.12

Business Process A can be automated by either Application A or Service Composition A. The delivery of Application A can result in a body of solution logic that is specific to and tailored for the business process. Service Composition A would be designed to automate the process with a combination of agnostic services and 40% of additional logic specific to the business process.

Reduced Volume of Logic Overall

The overall quantity of solution logic is reduced because the same solution logic is shared and reused to automate multiple business processes, as shown in Figure 4.13.

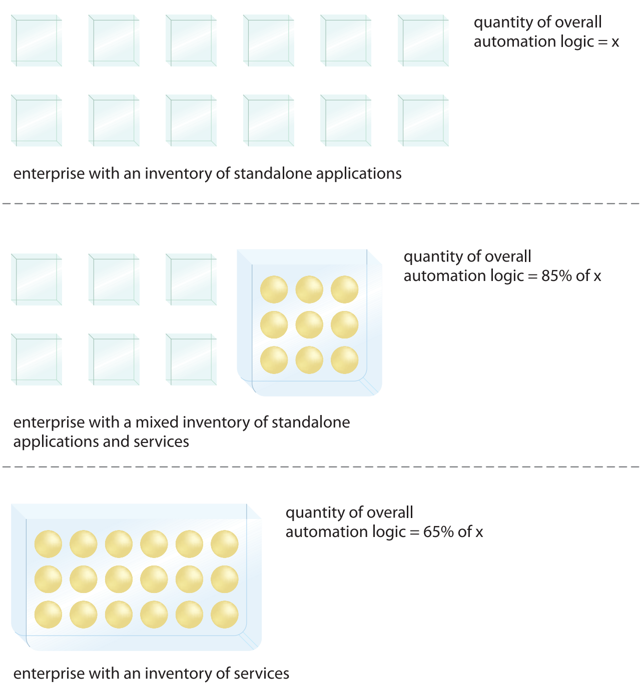


Figure 4.13

The quantity of solution logic shrinks as an enterprise transitions toward a standardized service inventory comprised of "normalized" services.

Inherent Interoperability

Common design characteristics consistently implemented result in solution logic that is naturally aligned. When this carries over to the standardization of service contracts and their underlying data models, a base level of automatic interoperability is achieved across services, as illustrated in Figure 4.14. (See also the *Service-Orientation and the Concept of "Integration"* section later in this chapter.)

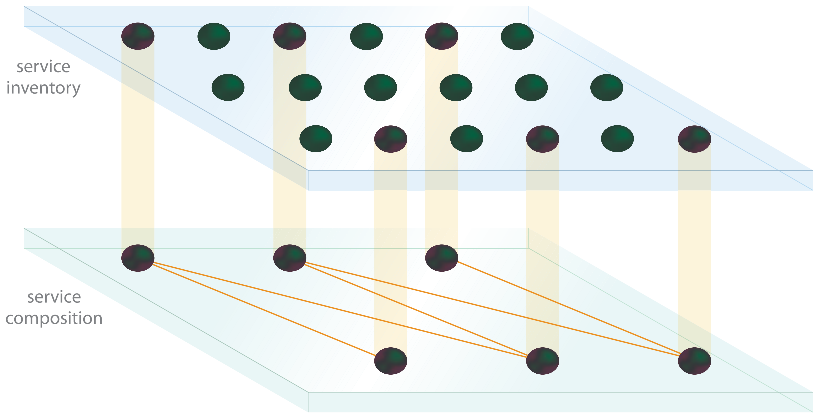


Figure 4.14

Services from different parts of a service inventory can be combined into new compositions. If these services are designed to be intrinsically interoperable, the effort to assemble them into new composition configurations is significantly reduced.

SUMMARY OF KEY POINTS

- The traditional silo-based approach to building applications has been successful at providing tangible benefits and measurable returns on investment.
- This approach has also caused its share of problems, most notably an increase in integration complexity and an increase in the size and administrative burden of IT enterprises.
- Service-orientation establishes a design paradigm that leverages and builds upon previous approaches and proposes a means of avoiding problems associated with silo-based application delivery.

4.3 Challenges Introduced by Service-Orientation

As much as service-orientation can solve some of the more significant historical problems in IT, its application in the real world can make some serious impositions. It is necessary to be aware of these challenges ahead of time because being prepared is key to overcoming them.

Design Complexity

With a constant emphasis on reuse, a significant percentage of a service inventory can ultimately be comprised of agnostic services capable of fulfilling requirements for multiple potential service consumer programs.

Although this can establish a highly normalized and streamlined architecture, it can also introduce an increased level of complexity for both the architecture as well as individual service designs.

Examples include:

- increased performance requirements resulting from the increased reuse of agnostic services
- reliability issues of services at peak concurrent usage times and availability issues of services during off-hours
- single point of failure issues introduced by excessive reuse of agnostic services (and that may require the need for redundant deployments to mitigate risks)
- increased demands on service hosting environments to accommodate autonomy-related preferences
- service contract versioning issues and the impact of potentially redundant service contracts

Design issues such as these can be addressed by a combination of sound technology architecture design, modern vendor runtime platform technology, and the consistent application of service-orientation design principles. Solving service reliability and performance issues in particular are primary goals of those design principles more focused on the underlying service logic, such as Service Autonomy, Service Statelessness, and Service Composability.

The Need for Design Standards

Design standards can be healthy for an enterprise in that they "pre-solve" problems by making several decisions for architects and developers ahead of time, thereby increasing the consistency and compatibility of solution designs. Their use is required in order to realize the successful propagation of service-orientation.

Although it can be a straight-forward process to create these standards, incorporating them into a (non-standardized) IT culture already set in its ways can be demanding to say the least. The usage of design standards can introduce the need to enforce their compliance, a policing role that can meet with resistance. Additionally, architects and developers sometimes feel that design standards inhibit their creativity and ability to innovate.

A circumstance that tends to aid the large-scale realization of standardization is when the SOA initiative is championed by an executive manager, such as a CIO. When an individual or a governing body has the authority to essentially "lay down the law," many of these cultural issues resolve themselves more quickly. However, within organizations based on peer-level departmental structures (which are more common in the public sector), the acceptance of design standards may require negotiation and compromise.

The best weapon for overcoming cultural resistance to design standards is communication and education. Those resisting standardization efforts are more likely to become supporters after gaining an appreciation of the strategic significance and ultimate benefits of adopting and respecting the need for design standards.

Top-Down Requirements

A preferred strategy to delivering services is to first conceptualize a service inventory by defining a blueprint of all planned services, their relationships, boundaries, and individual service models. This approach is very much associated with a top-down delivery strategy in that it can impose a significant amount of up-front analysis effort involving many members of business analysis and technology architecture groups.

Though preferred, achieving a comprehensive blueprint prior to building services is often not feasible. It is common for organizations to face budget and time constraints and tactical priorities that simply won't permit it. As a result, there are phased and iterative delivery approaches that allow for services to be produced earlier on. These, however, often come with trade-offs in that they can require the service designs to be revisited and revised at a later point. While this can introduce risks associated with the implementation of premature service designs, it is often considered an acceptable compromise.

The principles of service-orientation can be applied to services on an individual basis, allowing a reasonable degree of service-orientation to be achieved regardless of the approach. However, the actual quality of the resulting service designs is typically tied to how much of the top-down analysis work was completed prior to their delivery.

BEST PRACTICE

It is recommended that, at minimum, a high-level service inventory blueprint always be defined prior to creating physical service contracts. This establishes an important "broader" perspective in support of service-oriented analysis and service modeling processes and, ultimately, results in stronger and more durable service designs.

Counter-Agile Service Delivery in Support of Agile Solution Delivery

Irrespective of the potential top-down efforts needed for some SOA projects, the additional design considerations required to implement a meaningful measure of each of the eight design principles increases both the overall time and cost to deliver service logic.

This may appear contrary to the attention SOA has received for its ability to increase agility. To achieve the state of organizational agility described in Chapter 3 requires that service-orientation already be successfully implemented. This is what establishes an environment in which the delivery of solutions is much more agile.

However, given that it takes more initial effort to design and build services than it does to build a corresponding amount of logic that is not service-oriented, the process of delivering services in support of SOA can actually be *counter*-agile. This can cause issues for an organization that has tactical requirements or needs to be responsive while building a service inventory.

BEST PRACTICE

An effective approach, when sufficient resources are available, is to allow SOA initiatives to be delivered alongside existing legacy development and maintenance projects. This way, tactical requirements can continue to be fulfilled by traditional applications while the enterprise works toward a phased transition toward service-oriented computing.

Appendix B provides additional coverage of SOA delivery strategies that address tactical versus strategic service delivery requirements.

Governance Demands

The eventual existence of one or more service inventories represents the ultimate deliverable of the typical large-scale SOA initiative. A service inventory establishes a powerful reserve of standardized solution logic, a high percentage of which will ideally be classified as agnostic or reusable. Subsequent to their implementation, though, the management and evolution of these agnostic services can be responsible for some of the most profound changes imposed by service-orientation.

In the past, a standalone application was typically developed by a single project team. Members of this team often ended up remaining "attached" to the application for subsequent upgrades, maintenance, and extensions. This ownership model worked because the application's overall purpose and scope remained focused on the business tasks it was originally built to automate.

The body of solution logic represented by agnostic services, however, is intentionally positioned to *not* belong to any one business process. Although these services may have been delivered by a project team, that same team may not continue to own the service logic as it gets repeatedly utilized by other solutions, processes, and compositions.

Therefore, a special governance structure is required. This can introduce new resources, roles, processes, and even new groups or departments. Ultimately, when these issues are under control and the IT environment itself has successfully adapted to the required changes, the many benefits associated with this new computing platform are there for the taking. However, the process of moving to this new governance model can challenge traditional approaches and demand time, expense, and a great deal of patience.

SUMMARY OF KEY POINTS

- Applying service-orientation on a broad scale can introduce increased design complexity and the need for a consistent level of standardization.
- The construction of services can be expensive and time-consuming, introducing a more burdensome project delivery lifecycle, further compounded by some of the common top-down analysis requirements that may need to be in place before services can be built.
- Service inventory governance requirements can impose significant changes that can shake up the organizational structure of an IT department.

4.4 Additional Considerations

To supplement the benefits and challenges just covered, this section discusses some further aspects of service-orientation.

It Is Not a Revolutionary Paradigm

Service-orientation is not a brand new paradigm that aims to replace all that preceded it. It, in fact, incorporates and builds upon proven and successful elements from past paradigms and combines these with design approaches shaped to leverage recent technology innovations.

This is why we do not refer to SOA as a revolutionary model in the history of IT. It is simply the next stage in an evolutionary cycle that began with the application of modularity on a small scale (by organizing simple programming routines into shared modules for example) and has now spread to the potential modularization of the enterprise.

Enterprise-wide Standardization Is Not Required

There is a common misperception that unless design standardization is achieved globally throughout the entire enterprise, SOA will not succeed. Although design standardization is a critical success factor for SOA projects that is *ideally* achieved across an enterprise, it only needs to be realized to a meaningful extent for service-orientation to result in strategic benefit.

For example, service-orientation emphasizes the need for standardizing service data models to avoid unnecessary data transformation and other problematic issues that can compromise interoperability. The extent to which data model standardization is achieved determines the extent to which these problems will be avoided.

The goal is not always to eliminate problems entirely because that can be an unrealistic objective, especially in larger enterprises. Therefore, the goal is sometimes to just minimize problems by taking special considerations into account during service design.

In support of this approach, design patterns exist for organizing the division of an enterprise into more manageable domains. Data standardization is generally more easily attained within each domain, and transformation is then only required when exchanging data across these domains. Even though this does not achieve a global data model, it can still help establish a very meaningful level of interoperability.

Reuse Is Not an Absolute Requirement

Increasing reusability of solution logic is a fundamental goal of service-orientation, and reuse is clearly one of the most associated benefits of SOA. As a result, organizations that have had limited success with past reuse initiatives, or with concerns that significant amounts of reuse cannot be achieved within their enterprise, are often hesitant about SOA in general.

While reuse, especially over time, can be one of the most rewarding parts of investing in SOA, it is not the sole primary benefit. Perhaps even more fundamental to service-orientation than promoting reuse is fostering interoperability. Enabling an enterprise to connect previously disparate systems or to make interconnectivity an intrinsic quality of new solution logic is extremely powerful.

You could ignore the principle of Service Reusability in service designs and still achieve significant returns on investment based solely on raising the level of enterprise-wide interoperability.

NOTE

One could argue that reuse and interoperability are very closely related in that if two services are interoperable, there is always the opportunity for reuse. However, traditional perspectives of reusable solution logic focus on the nature of the logic itself. A service that is designed to be specifically agnostic to business processes and cross-cutting to address multiple concerns will have a particular functional context associated with it. Therefore, reuse can be seen as a separate design characteristic that relies and builds upon interoperability. See Chapter 9 for more details.

SUMMARY OF KEY POINTS

- Service-orientation has deep roots in several past computing platforms and design approaches, and is therefore not considered a revolutionary design paradigm.
- Global standardization within an enterprise is not a requirement for creating service-oriented enterprises because individual service inventories can be established (and separately standardized) within different enterprise domains.
- Although fundamental to much of service-orientation, if reusability were to be omitted as a design characteristic, significant interoperability-related benefit would still be attainable.

4.5 Effects of Service-Orientation on the Enterprise

There are good reasons to have high expectations from the service-orientation paradigm. But, at the same time, there is much to learn and understand before it can be successfully applied. The following sections explore some of the more common examples.

Service-Orientation and the Concept of "Application"

Having just stated that reuse is not an absolute requirement, it is important to acknowledge the fact that service-orientation does place an unprecedented emphasis on reuse. By establishing a service inventory with a high percentage of reusable and agnostic services, we are now positioning those services as the primary (or only) means by which the solution logic they represent can and should be accessed.

As a result, we make a very deliberate move away from the silos in which applications previously existed. Because we want to share reusable logic whenever possible, we automate existing, new, and augmented business processes through service composition. This results in a shift where more and more business requirements are fulfilled not by building or extending applications, but by simply composing existing services into new composition configurations.

When compositions become more common, the traditional concept of an application, a system, or a solution actually begins to fade, along with the silos that contain them. Applications no longer consist of self-contained bodies of programming logic responsible for automating a specific set of tasks (Figure 4.15). What was an application is now just another service composition. And it's a composition made up of services that very likely participate in other compositions (Figure 4.16).

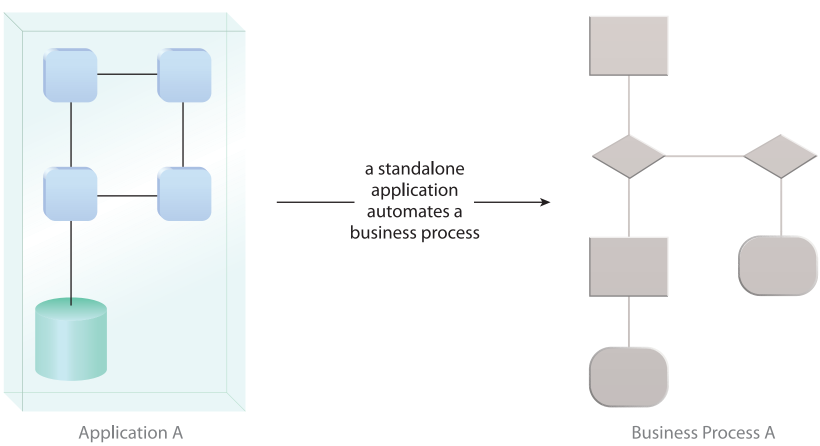


Figure 4.15
The traditional application, delivered to automate specific business process logic.

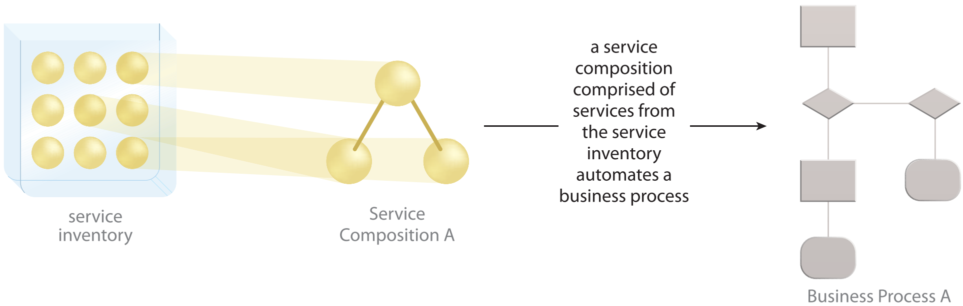


Figure 4.16

The service composition, intended to fulfill the role of the traditional application by leveraging agnostic and non-agnostic services from a service inventory. This essentially establishes a "composite application."

An application in this environment loses its individuality. One could argue that a service-oriented application actually does not exist because it is, in fact, just one of many service compositions. However, upon closer reflection, we can see that some of the services are actually not business process-agnostic. The task service, for example, intentionally represents logic that is dedicated to the automation of just one business task and therefore is not necessarily reusable.

What this indicates is that non-agnostic services can still be associated with the notion of an application. However, within service-oriented computing, the meaning of this term can change to reflect the fact that a potentially large portion of the application logic is no longer exclusive to the application.

Service-Orientation and the Concept of "Integration"

When we revisit the idea of a service inventory consisting of services that have, as per our service-orientation principles, been shaped into standardized and (for the most part) reusable units of solution logic, we can see that this can challenge the traditional perception of "integration."

In the past, integrating something implied connecting two or more applications or programs that may or may not have been compatible (Figure 4.17). Perhaps they were based on different technology platforms or maybe they were never designed to connect with anything outside of their own internal boundary. The increasing need to hook up disparate pieces of software to establish a reliable level of data exchange is what turned integration into an important, high profile part of the IT industry.

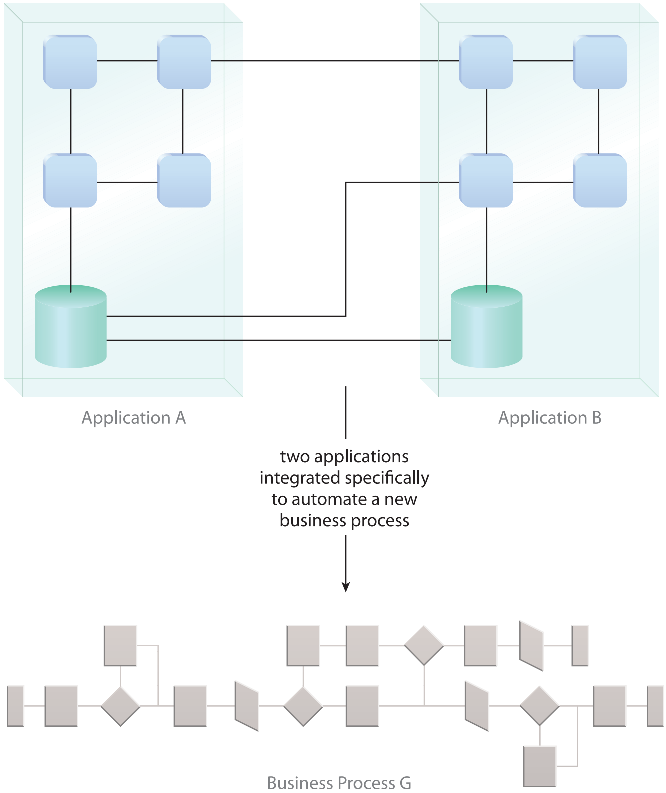


Figure 4.17

The traditional integration architecture, comprised of two or more applications connected in different ways to fulfill a new set of automation requirements (as dictated by the new Business Process G).

Services designed to be "intrinsically interoperable" are built with the full awareness that they will need to interact with a potentially large range of service consumers, most of which will be unknown at the time of their initial delivery. If a significant part of our enterprise solution logic is represented by an inventory of intrinsically interoperable services, it empowers us with the freedom to mix and match these services into infinite composition configurations to fulfill whatever automation requirements come our way.

As a result, the concept of integration begins to fade. Exchanging data between different units of solution logic becomes a natural and secondary design characteristic (Figure 4.18). Again, though, this is something that can only transpire when a substantial percentage of an organization's solution logic is represented by a quality service inventory.

While working toward achieving this environment, there will likely be many requirements for traditional integration between existing legacy systems and also between legacy systems and these services.

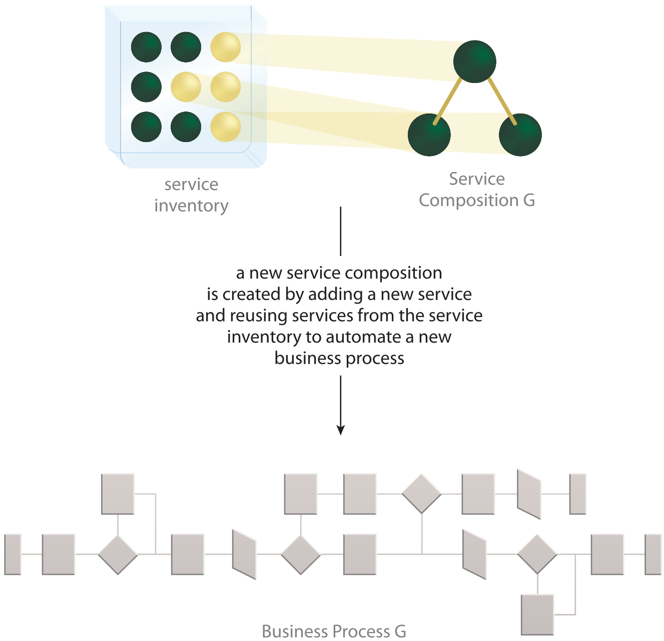


Figure 4.18

A new combination of services is composed together to fulfill the role of traditional integrated applications.

The Service Composition

Applications, integrated applications, solutions, systems, all of these terms and what they have traditionally represented can be directly associated with the service composition (Figure 4.19). However, given the fact that many SOA implementations consist of a mixture of legacy environments and services, these terms are sure to survive for quite some time.

In fact, as SOA transition initiatives continue to progress within an enterprise, it can be helpful to make a clear distinction between a traditional application (one which may reside alongside an SOA implementation or which may be actually encapsulated by a service) and the service compositions that eventually become more commonplace.

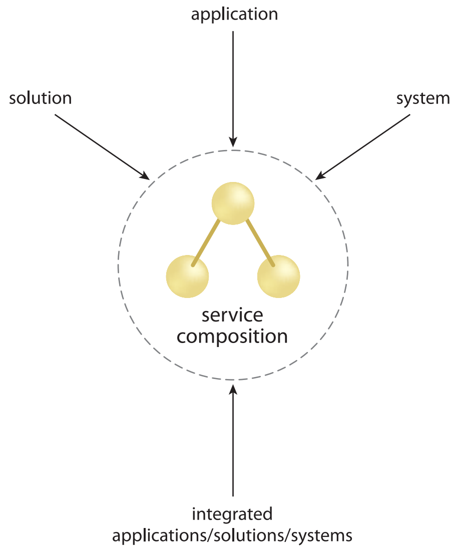


Figure 4.19
A service-oriented solution, application, or system is the equivalent of a service composition. If we were to build an enterprise-wide SOA from the ground up, it would likely be comprised of numerous service compositions capable of fulfilling the traditional roles associated with these terms.

Application, Integration, and Enterprise Architectures

Because applications have existed for as long as IT, when technology architecture as a profession and perspective within the enterprise came about, it made perfect sense to have separate architectural views dedicated to individual applications, integrated applications, and the enterprise as a whole.

When standardizing on service-orientation, the manner in which we document technology architecture is also in for a change. The enterprise-level perspective becomes predominant as it represents a master view of the service inventory. It can still encompass the traditional parts of a formal architecture, including conceptual views, physical views, and supporting technologies and governance platforms—but all these views are likely to now become associated with the service inventory.

A new type of technical specification that gains prominence in service-oriented enterprise initiatives is the *service composition architecture*. Even though we talk about the simplicity of combining services into new composition configurations on demand, it is by no means an easy process. It is a design exercise that requires the detailed documentation of the planned composition architecture.

For example, each service needs to be assessed as to its competency to fulfill its role as a composition member, and foreseeable service activity scenarios need to be mapped out.

Message designs, messaging routes, exception handling, cross-service transactions, policies, and many more considerations go into making a composition capable of automating its designated business process.

BEST PRACTICE

Although the structure and content of traditional application architecture specifications are augmented when documenting composition architectures, there can still be a natural tendency to refer to these documents as architecture specifications for applications.

While an organization is undergoing a transition toward SOA, it can be helpful to make a clear distinction between an application consisting of a service composition and traditional, standalone or legacy applications.

One approach is to consistently qualify the term "application." For example, it can be prefixed with "service-oriented," "composite," "standalone," or "legacy." Another option is to simply limit the use of the term "application" to refer to non-service-composed solutions only.

Furthermore, a composed service encapsulating a legacy application can be documented in separate specifications: a composition architecture specification that identifies the service and points to an application architecture specification that defines the corresponding application.

SUMMARY OF KEY POINTS

- The traditional concept of an application can change as more agnostic services become established parts of the enterprise.
- The traditional concept of integration can change as the proliferation of standardized, intrinsic interoperable services increases.
- Architectural views of the enterprise shift in response to the adoption of service-orientation. Principally, the enterprise perspective becomes increasingly prominent.

4.6 Origins and Influences of Service-Orientation

It is often said that the best way to understand something is to gain knowledge of its history. Service-orientation, by no means, is a design paradigm that just came out of nowhere. It is very much a representation of the evolution of IT and therefore has many roots in past paradigms and technologies (Figure 4.20). At the same time, it is still in a state of evolution itself and therefore remains subject to influences from on-going trends and movements.

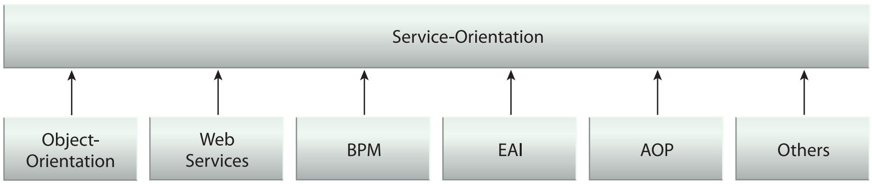


Figure 4.20
The primary influences of service-orientation also highlight its many origins.

The sections that follow describe some of the more prominent origins and thereby help clarify how service-orientation can relate to and even help further some of the goals from past paradigms.

Object-Orientation

In the 1990s the IT community embraced a design philosophy that would lead the way in defining how distributed solutions were to be built. This paradigm was object-orientation, and it came with its own set of principles, the application of which helped ensure consistency across numerous environments. These principles defined a specific type of relationship between units of solution logic classified as objects, which resulted in a predictable set of dynamics that ran through entire solutions.

Service-orientation is frequently compared to object-orientation, and rightly so. The principles and patterns behind object-oriented analysis and design represent one of the most significant sources of inspiration for this paradigm.

In fact, a subset of service-orientation principles (Service Reusability, Service Abstraction, and Service Composability, for example) can be traced back to object-oriented counterparts. What distinguishes service-orientation, though, are the parts of the object-oriented school of thought that were left out and the other principles that were added. See Chapter 14 for a comparative analysis of principles and concepts associated with these two design approaches.

Web Services

Even though service-orientation as a paradigm and SOA as a technology architecture are each implementation-neutral, their association with Web services has become commonplace—so much so that the primary SOA vendors have shaped their respective platforms around the utilization of Web services technology.

Although service-orientation remains a fully abstract paradigm, it is one that has historically been influenced by the SOA platforms and roadmaps produced by these vendors. As a result, the Web services framework has influenced and promoted several service-orientation principles, including Service Abstraction, Service Loose Coupling, and Service Composability.

Business Process Management (BPM)

BPM places a significant emphasis on business processes within the enterprise both in terms of streamlining process logic to improve efficiency and also to establish processes that are adaptable and extensible so that they can be augmented in response to business change.

The business process layer represents a core part of any service-oriented architecture. From a composition perspective, it usually assumes the role of the parent service composition controller. The advent of orchestration technology reaffirmed this role from an implementation perspective.

A primary goal of service-orientation is to establish a highly agile automation environment fully capable of adapting to change. This goal can be realized by abstracting business process logic into its own layer, thereby alleviating other services from having to repeatedly embed process logic.

While service-orientation itself is not as concerned with business process reengineering, it fully supports process optimization as a primary source of change for which services can be recomposed.

Enterprise Application Integration (EAI)

Integration became a primary focal point in the late 90's, and many organizations were ill prepared for it. Numerous systems were built with little thought given to how data could be shared outside of the system boundary. As a result, point-to-point integration channels were often created when data sharing requirements emerged. This led to well known problems associated with a lack of stability, extensibility, and inadequate interoperability frameworks.

EAI platforms introduced middleware that allowed for the abstraction of proprietary applications through the use of adapters, brokers, and orchestration engines. The resulting integration architectures were, in fact, more robust and extensible. However, they also became notorious for being overwhelmingly complex and expensive, as well as requiring long-term commitments to the middleware vendor's platform and roadmap.

The advent of the open Web services framework and its ability to fully abstract proprietary technology changed the face of integration middleware. Vendor ties could be broken by investing in mobile services as opposed to proprietary platforms, and organizations gained more control over the evolution of their integration architectures.

Several innovations that became popularized during the EAI era were recognized as being useful to the overall goals associated with building SOA using Web services. One example is the broker component, which allows for services using different schemas representing the same type of data to still communicate through runtime transformation. The other is the orchestration engine, which can actually be positioned to represent an entire service layer within larger SOA implementations. These parts of the EAI platform support several service-orientation principles, including Service Abstraction, Service Statelessness, Service Loose Coupling, and Service Composability.

Aspect-Oriented Programming (AOP)

A primary goal of AOP is to approach the separation of concerns with the intent of identifying specific concerns that are common to multiple applications or automation scenarios. These concerns are then classified as "cross-cutting," and the corresponding solution logic developed for cross-cutting concerns becomes naturally reusable.

Aspect-orientation emerged from object-orientation by building on the original goals of establishing reusable objects. Although not a primary influential factor of service-orientation, AOP does demonstrate a common goal in emphasizing the importance of investing in units of solution logic that are agnostic to business processes and applications and therefore highly reusable. It further promotes role-based development, allowing developers with different areas of expertise to collaborate.

NOTE

The actual events and timeline associated with the emergence of SOA are documented in Chapter 4 of the book *Service-Oriented Architecture: Concepts, Technology, and Design*.

SUMMARY OF KEY POINTS

- Service-orientation represents a design paradigm that has its roots in several origins. It emphasizes successful and proven approaches and supplements them with new principles that leverage recent conceptual and technology innovation.
- Service-orientation, as a design paradigm, is comparable with object-orientation. In fact, several key object-oriented principles have persisted in service-orientation.
- The Web services technology platform is primarily responsible for the popularity of SOA and is therefore also a significant influence in service-orientation. Conversely, the rise of service-oriented computing has repositioned and formalized the Web services technology set from its original incarnation.

4.7 CASE STUDY BACKGROUND

Cutit's immediate priority is to streamline their internal supply chain process. The order process in particular needs to be supported by the planned services so that orders and back-orders can be fulfilled as soon as possible.

Below are brief descriptions of the service candidates shown in Figure 4.21 in relation to how they inter-relate based on their entity-centric functional contexts:

- Everything originates with the manufacturing of chain blades in the Cutit lab, which requires the use of specific *materials* that are applied as per predefined *formulas*.
- The assembly of *chains* results in products being added to their overall *inventory*.
- *Saws* and *kits* are items Cutit purchases from different manufacturers to complement their chain models.
- *Notifications* need to be issued when stock levels fall below certain levels or if other urgent conditions occur.

- Finally, a periodic *patent sweep* is conducted to search for recently issued patents with similarities to Cutit's planned chain designs.

Note that all services shown are entity services, with the exception of Patent Sweep and Notifications, which are based on the utility service model. A task service is added in Part II.

Figure 4.21

The initial set of services planned to support the following types of processes: keeping track of orders and back-orders, chain manufacturing, tracking required manufacturing materials, and inventory management of manufactured and purchased products. All of the displayed services are based on the entity service model, except for the bottom two, which are utility services.

This page intentionally left blank

Index

E

F

T

X–Z