SERVICE-ORIENTED MODELING

SERVICE ANALYSIS, DESIGN, AND ARCHITECTURE

MICHAEL BELL

WILEY

JOHN WILEY & SONS, INC.

Service-Oriented Modeling

Service Analysis, Design, and Architecture

Michael Bell

WILEY

John Wiley & Sons, Inc.

This book is printed on acid-free paper.♾

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

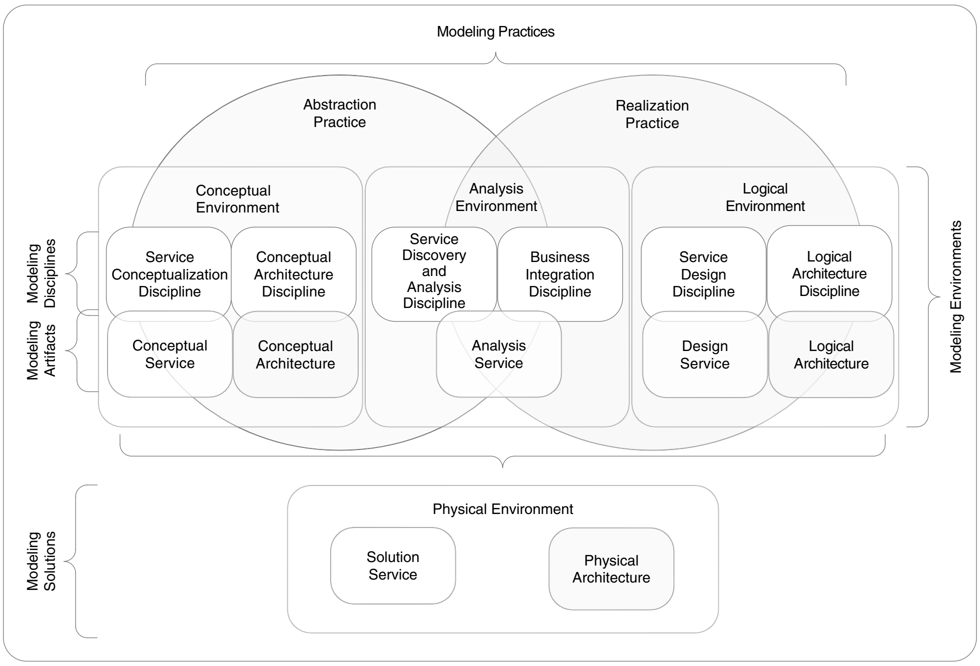
Bell, Michael, 1951—
Service-oriented modeling : service analysis, design, and architecture / Michael Bell.
p. cm.
Includes index.
ISBN 978-0-470-14111-3 (cloth)
1. Business enterprises—Computer networks—Management. 2. System
design. 3. Computer software—Development. 4. Computer simulation.
5. Computer network architectures. I. Title.
HD30.37.B45 2008
004.068—dc22

2007033463

Printed in the United States of America.

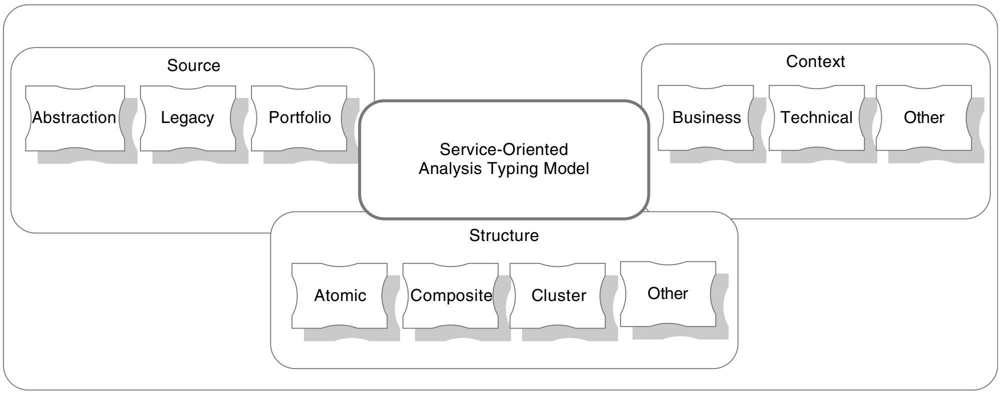
10 9 8 7 6 5 4 3 2 1

For Yvonne, whose love, patience, and support carried me through this project.



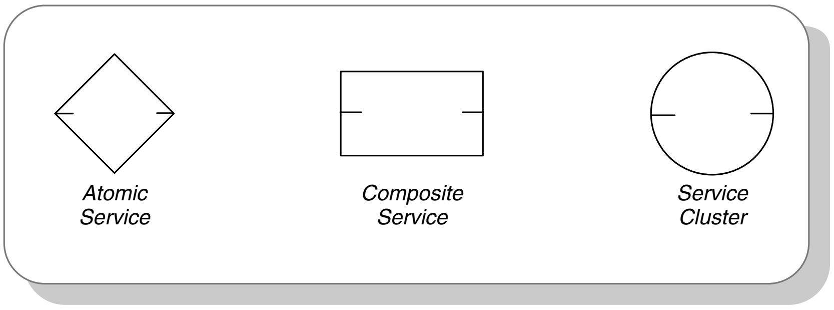
Service-Oriented Modeling Framework

Service Typing Model

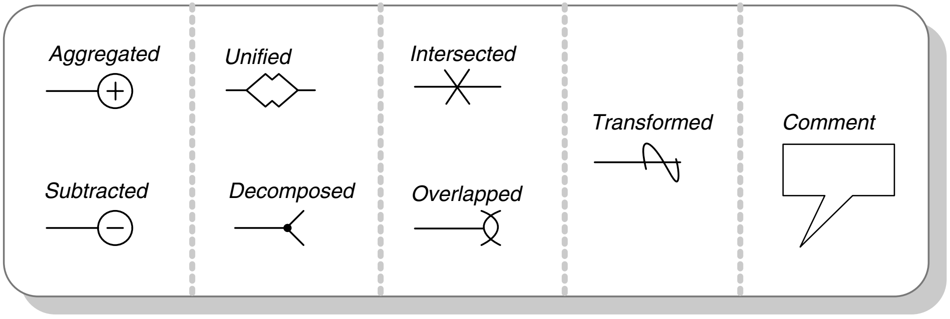


Service-Oriented Analysis Typing Model

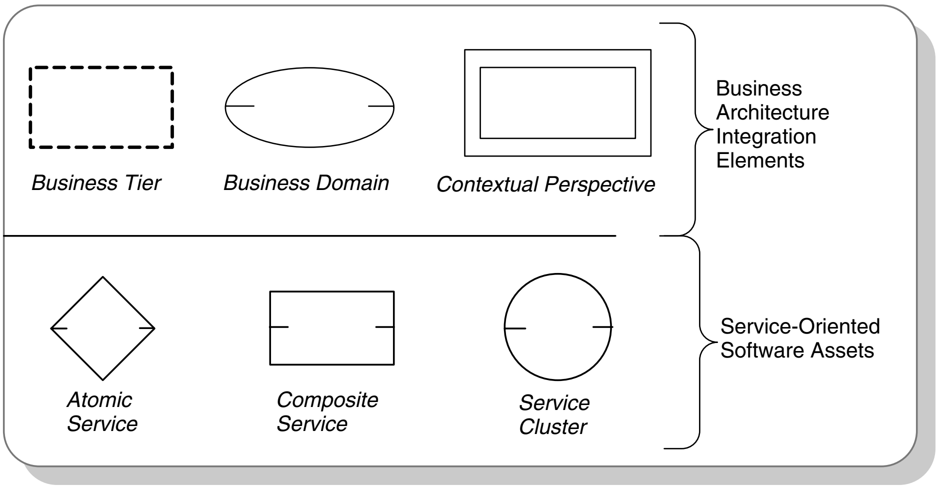
Service-Oriented Analysis Notation



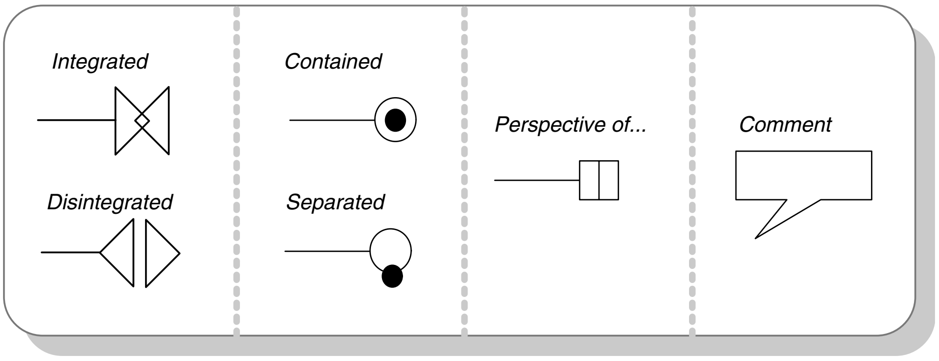
Service-Oriented Analysis Asset Notation



Service-Oriented Analysis Operation Notation

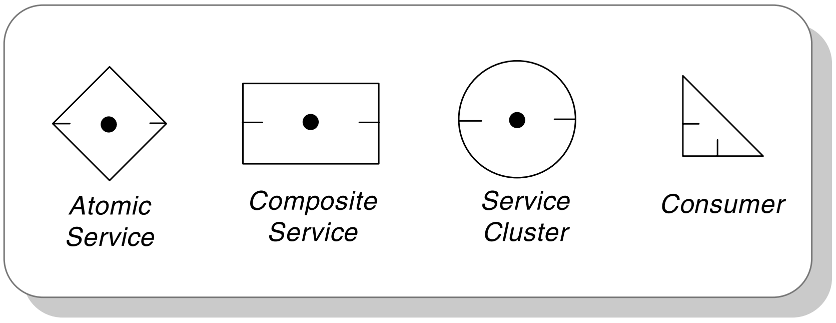


Service-Oriented Business Integration Asset Notation

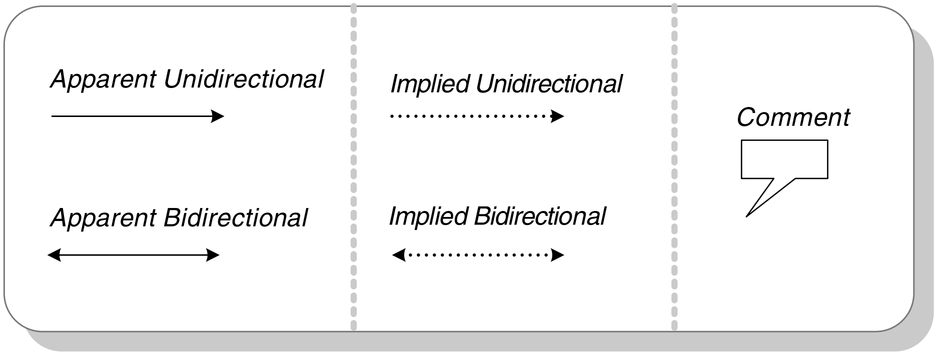


Service-Oriented Business Integration Operations Notation

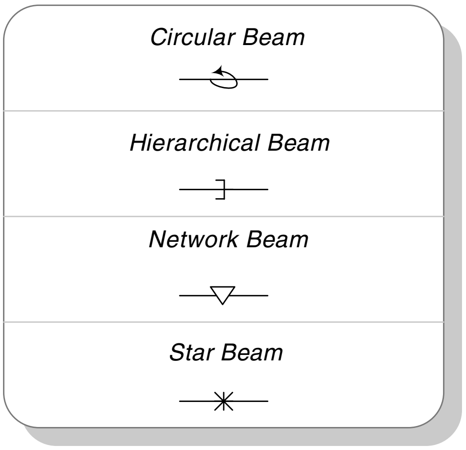
Service-Oriented Design Notation



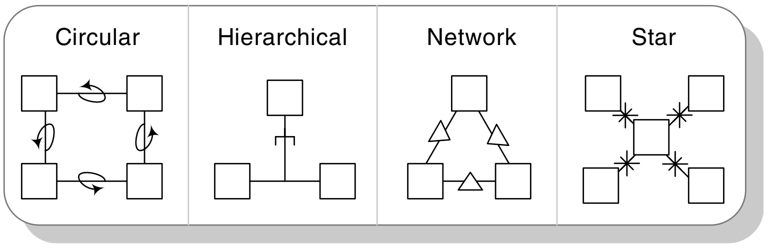
Service-Oriented Design Asset Notation



Service-Oriented Logical Design Relationship Connectors

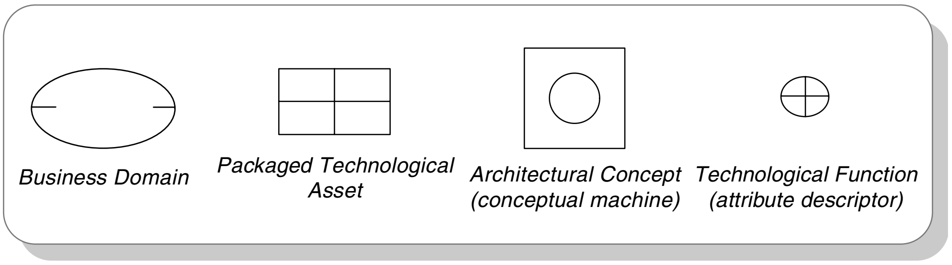


Service-Oriented Design Composition Style Beams

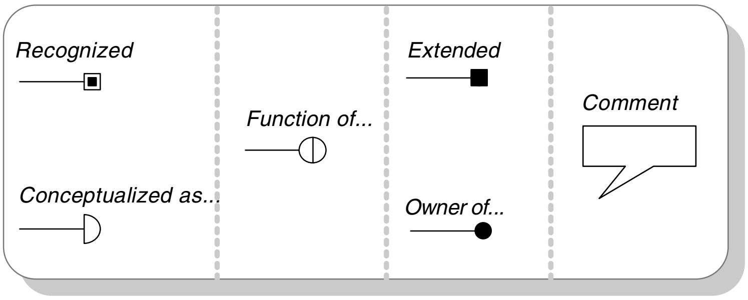


Logical Design Composition Styles

Service-Oriented Conceptual Architecture Notation

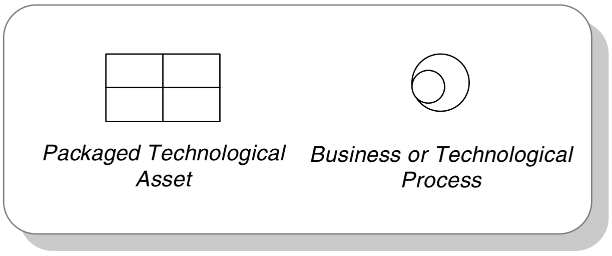

Business Domain
Packaged Technological Asset
Architectural Concept (conceptual machine)
Technological Function (attribute descriptor)

Conceptual Architecture Solution Elements

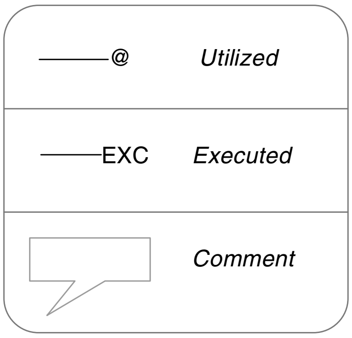


Conceptual Architecture Operations Notation

Service-Oriented Logical Architecture Notation


Packaged Technological Asset
Business or Technological Process

Logical Architecture Assets Notation



Logical Architecture Operations Notation

CONTENTS

PREFACE

Not long after the beginning of the current decade, the new service-oriented architecture (SOA) paradigm picked up steam and was established as a leading business and technology organizational concept. Lack of software asset reusability standards, absence of software interoperability disciplines, and incoherent business and technology strategies drove the enterprise to establish a more suitable model that promised to foster business agility and increase return on investment. This model also galvanized the development of SOA governance best practices, introduced SOA products, and promoted new service-oriented modeling disciplines.

The enterprise is still seeking mechanisms that can alleviate alignment challenges between business and information technology (IT) organizations. This effort includes the establishment of a common service taxonomy and vocabulary—an easy-to-understand language that can fill in the communication gaps between the problem and solution domain entities and establish a proper service development life cycle.

Unlike other SOA books on the market, this one introduces a service-oriented modeling framework that employs an agile and universal business and technology language to facilitate analysis, design, and architecture initiatives. The service-oriented modeling disciplines presented will enable practitioners to integrate existing legacy applications and to incorporate new ideas and concepts to address organizational concerns. These proposed best practices can be applied to all technologies, software platforms, and languages despite their physical location or ownership. Furthermore, business and IT professionals, such as managers, business analysts, business architects, technical architects, team leaders, and developers can now share the burden of software development initiatives as they are commissioned to bear equal responsibility and accountability.

The service-oriented modeling research presented in this book was driven by the following vision statements:

- Introduce a state-of-the-art and holistic modeling language that can facilitate an SOA implementation
- Introduce advanced service life cycle concepts and processes that can be employed to manage service-oriented projects
- Enable business and IT personnel to equally partner in service-oriented modeling efforts and to represent their unique perspectives

This book's mission focuses on the following service-oriented modeling disciplines that also offer an easy-to-understand modeling language and a notation that is simple to use:

- Service-oriented conceptualization
- Service-oriented discovery and analysis
- Service-oriented business integration
- Service-oriented design
- Service-oriented conceptual architecture
- Service-oriented logical architecture

Chapter 1 introduces the proposed service-oriented modeling framework and outlines its components. This chapter targets business and technology personnel who seek to establish and implement a service-oriented modeling language that can be employed during projects to guide, monitor, and control service development life cycles.

Part One discusses the service life cycle model and its various building blocks. It elaborates on service evolution management mechanisms during given projects and business initiatives. It also discusses various life cycle perspectives that enable monitoring and assessment of a project's process. Chapters 2 and 3 set the stage for the management of service-oriented modeling disciplines that are discussed throughout the book. They provide a solid framework for the service-oriented modeling environment and will be practical for business and IT executives, product managers, project managers, architects, and lead developers.

Part Two is dedicated to the service-oriented conceptualization process and elaborates on various mechanisms that can help organizations to establish common concepts and identify conceptual services and establish enterprise taxonomies. This Part is targeted at product managers, business architects, business analysts, technical architects, technical managers, team leaders, and developers. Chapters 4 and 5 introduce a step-by-step and an easy-to-employ concept discovery process that yields conceptual solution propositions to organizational problems.

Part Three delves into service-oriented discovery and analysis mechanisms. This Part furnishes best practices and procedures that should be used to discover new services and even employ legacy applications to provide viable business and technological solutions. Chapter 6 provides unique mechanisms to establish services' identities and to categorize them based on their distinctive characteristics. Chapter 7 enables product managers, business architects, technical architects, business analysts, technical leaders, and developers to perform service-oriented analysis on identified software assets. Chapter 8 introduces an analysis proposition modeling process that employs a service-oriented analysis language that can be further used in future service design, architecture, and construction phases.

Part Four depicts service-oriented business integration mechanisms and furnishes a business modeling language that can be used to integrate services with business domains and business products. Chapters 9, 10, and 11 expand on industry-standard business architecture and propose an implementation of business architecture disciplines. Business product managers, business managers, IT managers, business architects, technical architects, business analysts, and developers will find these chapters useful for alignment initiatives between business and technology organizations.

Part Five focuses on service-oriented design strategies, service relationships, logical compositions of services, and service behavior analysis. Chapters 12, 13, and 14 target analysis, architecture, and development personnel; these individuals must understand the nature of service-oriented software relationships as well as prepare packaged solutions for the architecture and construction teams, study service behaviors, and devise service-oriented transactions.

Finally, Part Six elaborates on fundamental aspects of service-oriented software architecture. These topics include conceptual and logical architecture modeling disciplines. Chapters 15 and 16 offer a conceptual architecture modeling language that can be employed to describe organizational technological abstractions, as well as a logical architecture topic that depicts the fundamental service-oriented building blocks that will be deployed to production and become an integral part of an organization's physical architecture.

ACKNOWLEDGMENTS

For all of those who have helped shape this book by contributing their ideas, vision and strategies, many thanks. Without their involvement, enthusiasm, and the coherent direction they provided, this book would not have been possible to accomplish.

The valuable insights and feedback that were extended through the research and the actual writing process were concerned with many fundamental aspects of business and technology practices. These include the general strategy of the book, business modeling aspects, business process disciplines, business architecture policies, enterprise and application architecture standards, computer programming topics, and even editorial contributions.

Charu Bansal, Donald Buckley, Dolly Dsa, Donald Mahaya, Eric Marks, Boris Minkin, Lisa Nathan, Mark Penna, Hormazd Pochara, Monica Roman, Jeff Schneider, and Alex Rozen. To all of you thank you for your time and commitment to the success of this book.

Special thanks to Mr. Sheck Cho, the executive editor that not only encouraged this project, but also provided strategic and tactical support throughout the publishing process.

CHAPTER 1

INTRODUCTION

As human beings, we are passionate about new ideas that promise to transform our lives and create new opportunities. We also tend to rapidly replace old technologies with new ones. Ours is a versatile society that runs on tomorrow's software piled on top of the technology layers of yesterday and today.

Try to imagine the next breakthrough that will supersede today's examples of human ingenuity. Will it be miniature software installed on microwave ovens or refrigerators that monitors a diet prescribed by a personal nutritionist? Could it be a smart software component that not only designs itself but also architects its own operating production environment? Or perhaps a virtual software development platform that enables business and technology personnel to jointly build applications with goggles and gloves?

These futuristic software concepts would probably contribute yet another layer to our already complex computing environments, one requiring resources to maintain and budgets to support. This layer would sit on top of technological artifacts accumulated over the past few decades that are already difficult to manage.

Not long after the new millennium, discontent over interoperability, reusability, and other issues drove the software community to come up with the service-oriented architecture (SOA) paradigm. Even readers who are not familiar with SOA will probably agree that it is rooted in traditional software development best practices and standards. To fulfill the promise of SOA, superb governance mechanisms are necessary to break up organizational silos and maximize software asset reusability. The SOA vision also addresses the challenges of tightly coupled software and advocates an architecture that relies on the loose coupling of assets. On the financial front, it tackles budgeting and return-on-investment issues. Another feature that benefits both the technological and business communities is a reduction of time to market and business agility. Indeed, the list of advantages continues to grow.

But does the promise of SOA address software diversity issues? Does it offer solutions to the integration and collaboration hurdles created by the accumulation of generations of heterogeneous computing landscapes? Will the SOA vision constitute yet another stratum of ideas and technologies that will be buried beneath future innovations? Will SOA be remembered as a hollow buzzword that failed to solve one of the most frustrating technological issues of our time? Or will it serve as an inspiration for generations to come?

It is possible that SOA may fail to deliver on its promise, but if it does, we, business and IT personnel, must shoulder some of the blame. SOA may turn out to be little more than a technological fire drill if we are ambivalent about the roles and responsibilities of legacy software in our existing and future organizational strategies; if we fail to tie together past, present, and future software development initiatives; and if we disregard the contributions of previous generations of architectures to today's business operations. Indeed, the idea of properly bridging new and old software technologies is a novel one. But what about establishing a more holistic view of the technological inventory that we have been building up for years? Can we treat all our software assets equally in terms of their analysis, design, and architectural value propositions? Can we understand their collaborative contribution to our environment without being too concerned about their underlying languages and implementation detail? Can we name these assets *services*? Can we conceive of them as *service-oriented* entities? Are they not built on similar SOA strategies and principles?

This book introduces service-oriented modeling mechanisms that will enable us to conceive software products that we have been constructing, acquiring, and integrating during the past few decades as *service-oriented* constituents. These entities—either legacy applications written in languages such as COBOL, PL1, Visual Basic, Java, C++, C#, or diverse empowering platforms and middleware—should all take part in an SOA modeling framework. Most important, they should be treated equally in the face of analysis, design, and architectural initiatives, and should simply be recognized as services.

A new SOA modeling language will be unveiled in this book that is *not* based on any particular programming language paradigm, constrained by language structure barriers, or limited to a language syntax. As a result of this universal language, the modeling process becomes more accessible to both the business and technology communities. This SOA modeling approach is well suited to provide tactical, short-term solutions to enterprise concerns, yet it furnishes strategic remedies to persistent organizational problems. So what is service-oriented modeling?

> Service-oriented modeling is a software development practice that employs modeling disciplines and language to provide strategic and tactical solutions to enterprise problems. This anthropomorphic modeling paradigm advocates a holistic view of the analysis, design, and architecture of all organizational software entities, conceiving them as service-oriented assets, namely services.

SERVICE-ORIENTED MODELING: WHAT IS IT ABOUT?

Modeling activities are typically embedded in the planning phase of almost any project or software development initiative that an organization conducts. The modeling paradigm embodies the analysis, design, and architectural disciplines that are being pursued during a given project. These major modeling efforts should not center only on design and architectural artifacts such as diagrams, charts, or blueprints. Indeed, modeling deliverables is a big part of a modeling process. But the service-oriented modeling venture is chiefly about simulating the real world. It is also about visualizing the final software product and envisioning the coexistence of services in an interoperable computing environment. Therefore, the service-oriented modeling paradigm advocates first creating a small replica of the "big thing" to represent its key characteristics and behavior—in other words, plan small, dream big; test small, execute big!

A VIRTUAL WORLD. How is it possible to simulate such a business and technological environment that offers solutions to organizational business and technology problems? "Simulating" does not necessarily mean starting with the construction of a software executable. It does not imply instantly embarking on an implementation initiative to produce source code and build components and services. The service-oriented modeling process begins with the construction of a miniature replica on paper. This may involve modeling teams in whiteboard analysis, design, and architecture sessions, or even the employment of software modeling tools that can visually illustrate the solutions arrived at. Thus, the simulation process entails the creation of a virtual world in which software constituents interface and collaborate to provide a viable remedy to an organizational concern.

A STRATEGIC ENDEAVOR GUIDED BY MODELING DISCIPLINES. Creating a miniature mockup of a final software product and its supporting environment can obviously reduce investment risk by ensuring the success of the impending software construction initiative. This can be achieved by employing analysis, design, and architectural disciplines that are driven by a modeling strategy that fosters asset reusability, a high return on investment, and a persuasive value proposition for the organization.

Service-oriented modeling disciplines enable us to focus on modeling strategies rather than being concerned with source code and detailed programming algorithms. By employing this modeling paradigm we raise the bar from the granular constructs of our applications, yet we must accommodate the language requirements of the underpinning platforms. We focus on identifying high-level business and technological asset reusability and consolidation opportunities, but we must also foster the reuse of software building blocks such as components and libraries. We rigorously search for interoperability solutions that can bridge heterogeneous technological environments, but we also concentrate on integration and message exchange implementation detail.

A LEARNING AND VALIDATION PROCESS. By producing a small version of the final artifact we are also engaging in a *learning* and verification process. This activity characteristically would enable us to validate the hypothesis that we have made about a software product's capability and its ability to operate flawlessly later on in a production environment. We are also being given the opportunity to inspect key aspects of software behavior, examine the relationships between software components, and even understand their internal and external structures. We are involved in a software assessment process that validates the business and technological motivation behind the construction of our tangible services.

To better understand the key characteristics of a future software product and its environment, the assessment effort typically leads to a proof-of-concept, a smaller construction project that concludes the service-oriented modeling initiative. This small-scale software executable, if

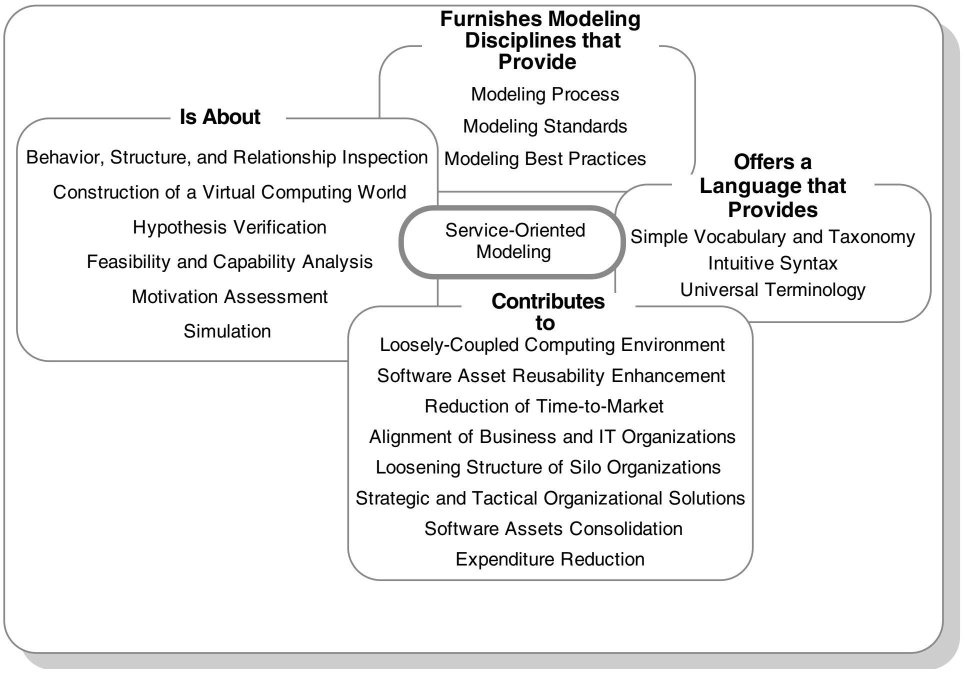

Furnishes Modeling Disciplines that Provide
Modeling Process
Modeling Standards
Modeling Best Practices
Is About
Behavior, Structure, and Relationship Inspection
Construction of a Virtual Computing World
Hypothesis Verification
Feasibility and Capability Analysis
Motivation Assessment
Simulation
Service-Oriented Modeling
Offers a Language that Provides
Simple Vocabulary and Taxonomy
Intuitive Syntax
Universal Terminology
Contributes to
Loosely-Coupled Computing Environment
Software Asset Reusability Enhancement
Reduction of Time-to-Market
Alignment of Business and IT Organizations
Loosening Structure of Silo Organizations
Strategic and Tactical Organizational Solutions
Software Assets Consolidation
Expenditure Reduction

EXHIBIT 1.1 ESSENCE OF THE SERVICE-ORIENTED MODELING PARADIGM

approved and agreed on, can later serve as the foundation for the ultimate service construction process.

EVERYBODY'S LANGUAGE: A UNIVERSAL LANGUAGE FOR BUSINESS AND TECHNOLOGY. We are often driven by tactical decisions to alleviate organizational concerns and to provide rapid solutions to problems that arise. The proposed service-oriented modeling language is designed to ease time to market by strengthening the ties between business and technology organizations. This can accelerate the delivery process of software assets to the production environment. Furthermore, the service-oriented modeling language can also be employed to fill in communication gaps and enhance alignment between the problem and solution domain bodies.

To achieve these goals, the service-oriented modeling language offers an intuitive syntax, a simple vocabulary, and a taxonomy that can be well understood and easily employed by various business and technology stakeholders during service life cycle phases and projects. The language can be utilized not only by professional modelers, architects, or developers, but by managers, business executives, business analysts, business architects, and even project administrators.

Exhibit 1.1 depicts the service-oriented modeling activities, language, disciplines, and benefits.

DRIVING PRINCIPLES OF SERVICE-ORIENTED MODELING

Service-oriented modeling principles capitalize on devised SOA standards already in use by organizations and professionals. These are best practices that are designed to foster strategic solutions to address enterprise concerns, and to overcome the shortsightedness that is frequently attributed to organizational tactical decisions. The following modeling principles promote business agility, software asset reuse, loosely coupled service-oriented environments, and a universal modeling language that can address software interoperability challenges:

- Virtualization
- Metamorphosis
- Literate modeling

VIRTUALIZATION. Modeling software is essentially a process of manipulating intangible entities. These are typically nonphysical assets that reside in peoples' minds or appear on paper. An effective modeling process should be as visual as possible, enabling business and technology personnel to view software elements as if they were concrete assets.

The virtuality and reality aspects of our surroundings have been debated by numerous philosophers going back to the eighteenth century. The traditional assertions that "everything has a reality and a virtuality" or "everything other than what is virtual is reality" are in agreement with sociologist, philosopher, and information technology pioneer Ted Nelson's claim that virtuality is the focal point of software design.[1] He further argued that virtuality is about designing software conceptual structure and feel.[2]

The visual aspect of the service-oriented modeling paradigm is driven by the construction of a virtual world in which elements seem almost as tangible as real physical objects. This world that we create to simulate reality is made up of two major elements: (1) the landscape that "glues" all pieces together; meaning the environment that empowers and executes services and (2) the services that communicate, interact, and exchange information to provide business value. Moreover, a virtual world can effectively simulate a heterogeneous computing landscape by treating software assets as equal partners in a modeling endeavor. This effect is called federated modeling.

The visualization process that we pursue enables us to model relationships, structures, and behaviors of services that would provide satisfying solutions to organizational problems. These goals can be achieved by fostering asset reusability, promoting a loosely coupled computing environment, and resolving interoperability challenges across organizations.

METAMORPHOSIS. The business environment that we all share is a dynamic market that keeps evolving and changing direction and also influences technological trends and application development. This vibrant business landscape often dictates alterations to a service's behavior, structure, and relationship to its environment during its life span. These modifications typically start at a service's inception, when it manifests as an intangible entity—an idea—and then continue as the service evolves into a physical software asset that executes business functionality in production. This transformation process is the essence of the metamorphosis paradigm driven by service-oriented modeling disciplines that ensure software elasticity, and ultimately, business agility.

But the transformation process does not stop with the deployment of services to production. Imagine how frequently a valuable application is involved in multiple project iterations that lead to software upgrades. Think about the myriad instances of a service finding its way back to the drawing board to be redesigned and then ferried back to the production environment again. This is typical of the software development life cycle, during which software products are upgraded, enhanced, and redistributed.

LITERATE MODELING. Should a programming language offer a simple syntax that is easy to understand, and is intuitive and readable? Or should it offer formal grammar, rules, and symbols that only developers can handle? Should a programming language be based on machine-readable source code that is easy to debug and optimize? Or should it be considered a scientific artifact, a mathematical formula that only experts on the subject can deliver?

This long-running debate is believed to have begun in the early 1980s and encompasses two major approaches to software development that have major ramifications for the service-oriented modeling paradigm. The first was introduced by Donald Knuth's theory of "literate programming" in which he argues: "I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: 'Literate Programming'."

The second approach was introduced by Edsger Dijkstra in his 1988 article "On the Cruelty of Really Teaching Computer Science," in which he claims that programming is merely a branch of mathematics.[3] He writes: "Hence, computing science is—and will always be—concerned with interplay between mechanized and human symbol manipulation, usually referred to as 'computing' and 'programming' respectively."

This debate further informs the discussion about the modeling paradigm. Should service-oriented modeling be founded on a specific programming platform structure that only developers and modelers can utilize? Or should modeling disciplines offer universal and easy to understand notations that are independent of language? Should a service-oriented modeling approach be tied to fashionable technologies? Or should a modeling language offer tools to design and architect multiple generations of legacy platforms, applications, and middleware?

The service-oriented anthropomorphic modeling approach provides easy mechanisms to address analysis, design, and architectural challenges and perceives software assets as having human characteristics. In the virtual world that we are commissioned to create, services "interact," "behave," "exchange information," and "collaborate;" they are "retired," "promoted," "demoted," and "orchestrated." Inanimate software entities are often treated as though they had human qualities. This "literate modeling" approach obviously enhances the strategies that are pursued during business initiatives and projects.

ORGANIZATIONAL SERVICE-ORIENTED SOFTWARE ASSETS

The service-oriented modeling paradigm regards all organizational software assets as candidates for modeling activities. We not only conceive them as our service-oriented modeling elements, meaning *services*, but we also evaluate them based on their contribution to a service-oriented environment, in terms of integration, collaboration, reusability, and consumption capabilities. These assets are also subjected to the modeling discipline activities depicted throughout this book. They are the enduring artifacts of the service-oriented modeling process and are regarded as units of concern, discovery, analysis, design, and architecture in a business initiative or a service-oriented project. Exhibit 1.2 illustrates the various service-oriented software assets that can be involved in providing solutions to organizational concerns: concepts, foundation software, legacy software, repositories, and utility software.

ORGANIZATIONAL CONCEPTS. Business or technical concepts embody an organization's formalized ideas, which are regarded as components of propositions to organizational concerns. These abstractions typically capture enterprise problems and offer remedies to alleviate negative effects on business execution. Concepts characteristically offer direction and strategy to the service-oriented analysis, discovery, design, and architectural disciplines. They also contribute to the establishment of a common organizational business and technical terminology that can be employed to fill in the communication gaps between business and information technology (IT) organizations. Agriculture Community Center, Business Community, and Pals' Community are examples of concepts that identify the financial prospects and marketing targets of a business goal. Here, the Community business concern is the driving aspect behind a particular organization's culture and strategy.

FOUNDATION SOFTWARE. Organizational empowering middleware and platform products are the basic software ingredients of the service-oriented modeling practice. Middleware products offer integration, hosting, and network environment support, including message orchestration and routing, data transformation, protocol conversion, and searching and binding capabilities. This software asset category may include application servers, portal products, software proxies,

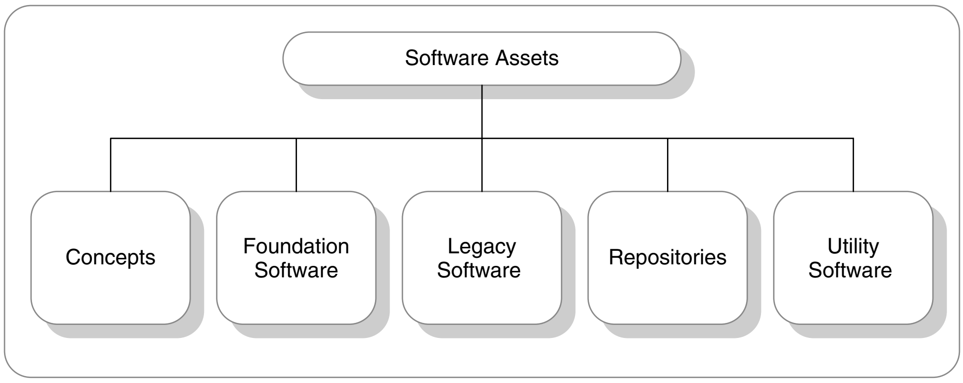


EXHIBIT 1.2 ORGANIZATIONAL SERVICE-ORIENTED SOFTWARE ASSETS

SOA intermediaries, gateways, universal description, discovery and integration (UDDI) registries, and even content management systems. Message-oriented middleware (MOM) technologies are also regarded as middleware, which may include traditional message busses or enterprise service busses (ESBs). Conversely, software platforms are akin to frameworks that enable languages to run. This category may also include operating systems, runtime libraries, and virtual machines.

LEGACY SOFTWARE. “Legacy” refers to existing software assets that are regarded as applications. These include business and technology software executables that already operate in the production environment. The term “legacy” also includes deployed organizational services such as Web services and even services that are running on a mainframe or other platforms and do not comply with Web service technologies and standards. Third-party vendor applications, service consumers, and partner services that operate outside of an organization are also conceived as legacy software assets. Customer Profile Service, Accounts Payable Application, or Trading Consumer are examples of existing and operating legacy software products, offered by third-party vendors or custom built by an organization’s internal development personnel.

REPOSITORIES. Repositories play a major role in most service-oriented modeling activities. This software category is characteristically provided by third-party vendor products that require organizational adoption and integration policies. These are software entities that offer storage facilities, such as relational databases, data warehouse repositories, and various database storage management products, such as data optimization and replication. The repository category can also include data-about-data, meaning repositories that do not necessarily store the actual data but describe it. These are known as meta-data repositories. We employ these storage facilities for a variety of management and data organization purposes. For example, meta-data repositories are used for governance rules, service life cycle management, security policies, search categories, and document management.

SOFTWARE UTILITIES. Utility executables are typically regarded as nontransactional software assets employed to facilitate flawless system operations in a production environment. These utilities chiefly offer performance-monitoring services, enforce service-level agreements (SLAs) between consumer and producers, track security infringements, and provide alert mechanisms in case of contract violations or system intrusions. Moreover, software utilities also provide provisioning and asset portfolio management facilities and even message mediation policy management between message exchange parties.

SERVICE-ORIENTED MODELING PROCESS STAKEHOLDERS

The service-oriented modeling process is characteristically overseen by SOA governance and SOA Center of Excellence enterprise bodies that provide best practices, standards, guidance, and assistance to service-oriented modeling activities. Moreover, the modeling process involves two major stakeholders, the problem and the solution domain organizations, typically managed by business and IT personnel that partner in an array of business and technological initiatives, such as a small project or a large service life cycle venture that may include a number of smaller projects.

The involvement of two major stakeholder groups is anticipated, each of which represents a different perspective; they are both vital contributors to a service-oriented modeling effort. In addition, they must collaborate and jointly facilitate alignment between business and IT organizations. Exhibit 1.3 illustrates these two major views that collaboratively contribute to the service-oriented modeling process: the business view and the technological view. Note the overseeing governance and Center of Excellence bodies that drive modeling activities.

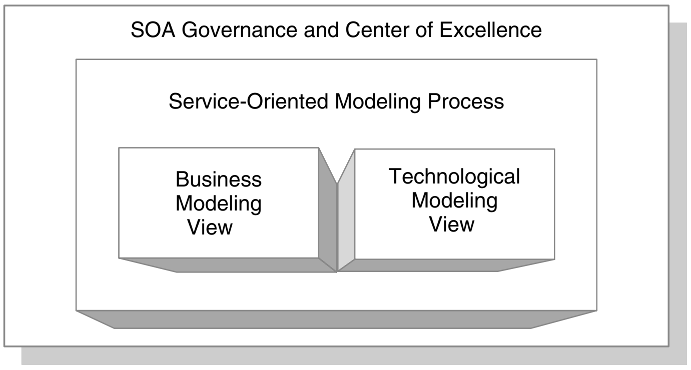

SOA Governance and Center of Excellence
Service-Oriented Modeling Process
Business Modeling View
Technological Modeling View

EXHIBIT 1.3 SERVICE-ORIENTED MODELING PERSPECTIVES

BUSINESS STAKEHOLDERS MODELING VIEW. This perspective represents business personnel who not only understand the financial implications of a project or a larger business initiative but have mastered the various business processes of an organization. They typically elaborate on the problem domain, present business requirements, and propose business solutions. Business professionals, however, should be equal participants in the discovery, analysis, design, and architecture modeling sessions. They should be familiar with the service-oriented modeling language and able to provide valuable input to the resulting modeling artifacts. The business organization can be represented by product managers, business managers, financial analysts, business analysts, business architects, business modelers, and even top-level executives, such as chief information officers (CIOs) or chief technology officers (CTOs).

TECHNOLOGY STAKEHOLDERS MODELING VIEW. The IT organization is intrinsically engaged in the technological aspects of the service-oriented modeling process. IT personnel contribute both to the analysis and design aspects of services and to the various architecture modeling activities. These functions include the establishment of asset integration strategies, consumption and reusability analysis, and service deployment planning. The IT organization must also ensure alignment with the organizational business model, business strategies, and business requirements. The technology perspective should be represented by technical management, application architects, system analysts, developers, service modelers, data modelers and database architects.

MODELING SERVICES INTRODUCTION: A METAMORPHOSIS EMBODIMENT

One of the most common questions people ask themselves when they are commissioned to provide a solution to an organizational problem is "What should be modeled?" The modeling world constitutes a blend of old and new software assets, ideas, formulated concepts, processes, and even people. Typically a remedy is being sought to an organizational concern that involves a thorough analysis of the existing physical operating environments and also requires the integration of new propositions to form viable business and technological solutions. This brings us to the understanding that service-oriented assets—whether they are abstractions, legacy applications, middleware, or software platforms—must be both conceived as services and categorized according to their role in the modeling process.

Which standard should be used to classify modeling services? The answer to this question is rooted in the necessity to establish a modeling language—a vocabulary—along with disciplines

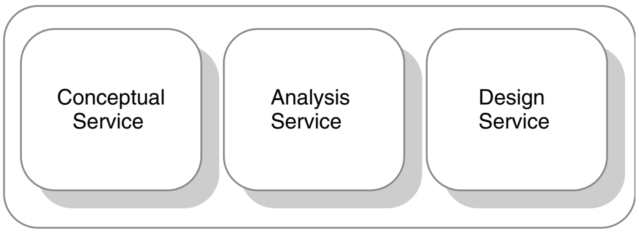


EXHIBIT 1.4 MODELING SERVICES

that can be utilized to implement service-oriented modeling tasks. Thus, the service-oriented modeling paradigm treats services according to their life cycle state and their corresponding disciplines. We thus propose three distinct categories: conceptual service, analysis service, and design service (see Exhibit 1.4). Consequently, the conceptual service originates during the service-oriented conceptualization phase; the analysis service is treated during the service-oriented discovery and analysis process, and the design service is employed during the service-oriented design stage. The driving disciplines behind these modeling processes are further discussed in the Service-Oriented Modeling Disciplines: Introduction section of this chapter.

CONCEPTUAL SERVICE: AN ABSTRACTION. A modeling process must offer a communication language, provide a distinct vocabulary that different stakeholders can use to collaborate and interface, to establish an organizational taxonomy that can be easily learned and enhanced as time goes by, and to support a terminology that depicts business and technological abstractions. These generalized idioms embody the requirements to provide solutions to an enterprise concern while reflecting the approach to solving a problem. We conceive these abstractions as conceptual services that are an essential deliverable in the service-oriented conceptualization phase. For example, the data aggregator concept can be regarded as a conceptual solution that aims to solve information collection problems that occur in an enterprise Web portal. The commission calculator is another conceptual service that exemplifies an essential solution to a business requirement to enable commission calculations for stockbrokers in an equity trading system.

Where does a conceptual service originate from? An undocumented idea or an informal enterprise proposal to solve a problem can be established as a conceptual service. These concepts can simply be expressed in meetings or whiteboard design sessions. A business process can also be regarded as a conceptual service candidate. A more formalized process, however, that takes place during service conceptualization facilitates the identification of new concepts derived from business and technological requirements (see Chapters 4 and 5 for a complete service conceptualization process).

Consider the following major attributes of a conceptual service: It

- Embodies business or technical context.
- Must be elastic enough to accommodate future business changes.
- Should focus on a solution rather than propose remedies to a wide range of problems, and should avoid business or technological context ambiguity.
- Corresponds to a business or technological requirement and depicts a coherent proposition.
- Represents a business or technological abstraction that can be added to an organization's language dictionary.
- Contributes to an organizational business or technological taxonomy.

ANALYSIS SERVICE: A UNIT OF ANALYSIS. A modeling process must offer a platform on which solution propositions to organizational concerns are verified for their viability and capacity to solve problems; enable proper validation of the assumptions that business and technology personnel make to address business requirements; and permit further analysis of the supporting services that take part in a solution, a project, or a business initiative. During the service-oriented discovery and analysis phase, we are allowed to test service collaboration and conduct further experiments in the search for the best possible offered resolution. An analysis service is the vehicle that enables us to reexamine these preliminary proposed remedies that were brought to the table in the first place.

But where does an analysis service originate from? The rule of thumb suggests that all services that participate in the analysis process should be regarded as analysis entities. In other words, all service-oriented assets are conceived of as units of analysis because of their involvement in a solution proposition during the analysis phase. To read more about the service-oriented discovery and analysis process, refer to the Service-Oriented Modeling Disciplines: Introduction section in this chapter, or the service-oriented discovery and analysis method discussed in Chapters 6, 7, and 8.

Consider the following major attributes of an analysis service. It

- Represents tangible (legacy software) or intangible (abstraction) service-oriented assets that participate in a solution.
- Can be associated with business or technical context.
- Must present a lucid internal structure.
- Should be composed of service-oriented software assets that can be decomposed during the service-oriented analysis modeling process to achieve a loose-coupling architectural effect.
- Or, should have a flexible internal structure that would allow aggregation of external services during the analysis modeling process.

DESIGN SERVICE: A LOGICAL SOLUTION PROVIDER AND A CONTRACTUAL ENTITY. All service-oriented assets that take part in a design process can be regarded as design services. A design service is a modeling element that enables us to visualize and plan future service behavior, structure, and peer relationships in a production environment. This service-oriented design asset, whose collaborators are peer services and consumers—bound by a stipulated contract—participates in a group effort to provide a viable design solution to a business or technological problem. To achieve this goal, we primarily focus on the message and information exchange capabilities of a service. This would contribute to its future capacity to interact and collaborate with its surrounding environment, and most important, to its ability to abide by the service level agreement (SLA) that it is committed to. This scheme is called a *logical solution*, and the design service that participates in this venture is known as *logical solution provider*.

A design service must also be a part of a service orchestration initiative, during which we coordinate and synchronize service functionality to enable flawless message exchange and efficient transaction execution. This logical design effort would ensure the quality of service offerings and contribute to the creation of a harmonized interactive environment.

Consider the following major attributes of a design service; it

- Can be associated with business or technical context.
- Can represent a tangible (legacy software) or an intangible (abstraction) service-oriented asset.
- Should be interfaceable, containing one or more interfaces to be utilized by potential consumers.