CHAPTER 1

INTRODUCTION

COPYRIGHTED MATERIAL

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

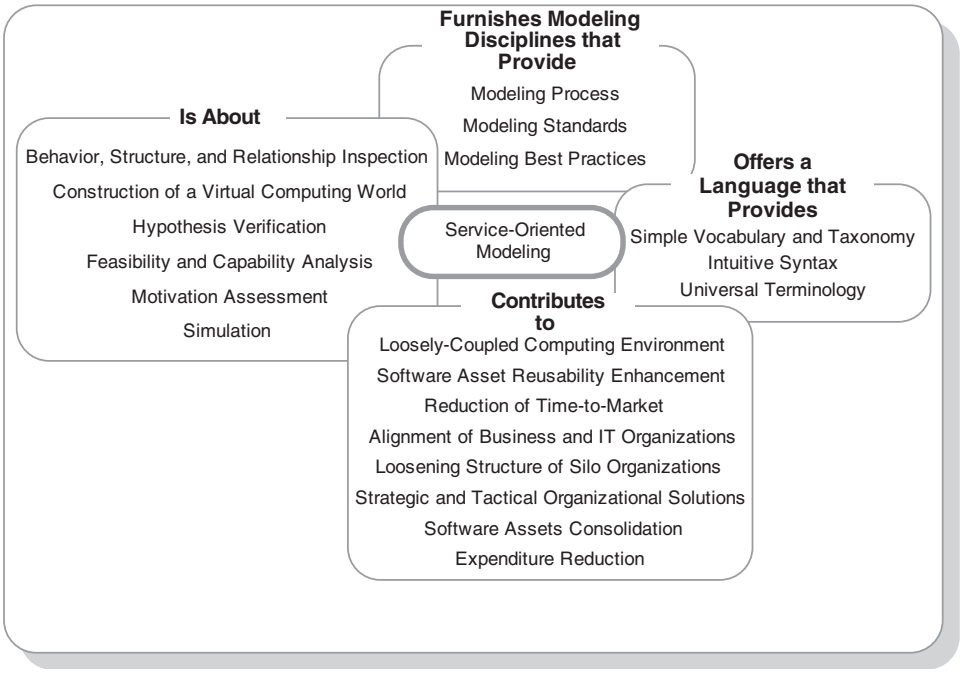

EXHIBIT 1.1 ESSENCE OF THE SERVICE-ORIENTED MODELING PARADIGM

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,

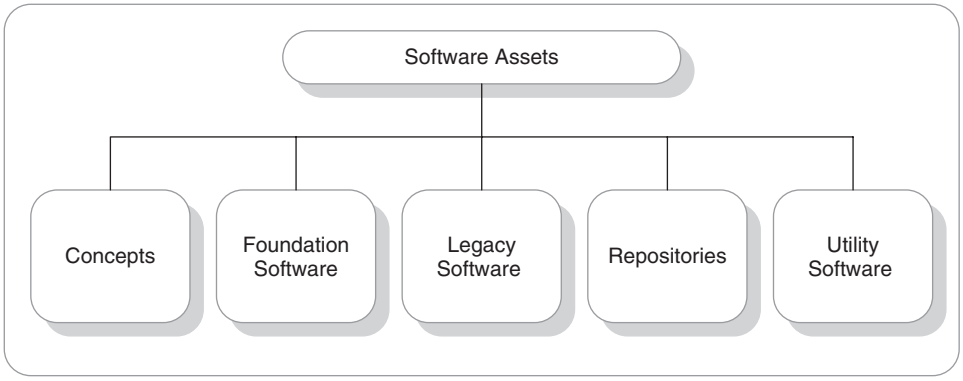

EXHIBIT 1.2 ORGANIZATIONAL SERVICE-ORIENTED SOFTWARE ASSETS

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 buses or enterprise service buses (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.

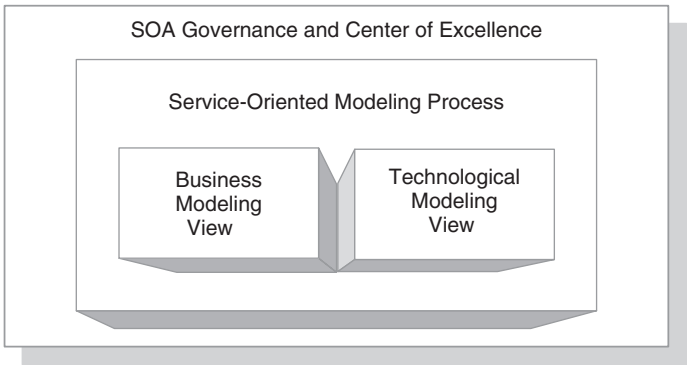

EXHIBIT 1.3 SERVICE-ORIENTED MODELING PERSPECTIVES

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, technical 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

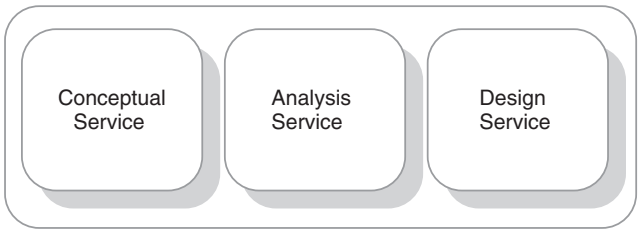

EXHIBIT 1.4 MODELING SERVICES

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.

- Is bound by a service-oriented contract that depicts its interfaces and other important requirements, such as accessibility, consumption, reusability, and security.
- Must participate in an orchestration design initiative.

SOLUTION SERVICE: PHYSICAL SOFTWARE ASSET. A solution service is a tangible software asset, constructed by business and technology development teams, that has concluded its development life cycle and is deployed and integrated in a production environment. This asset may be a custom-built software product designed in house, an acquired third-party vendor solution package, or a component custom built by an outsourcing firm. It is the ultimate deliverable of a software development process. Regardless of its origin, we typically employ a solution service to participate in a business or a technological solution devised by the service-oriented modeling practice. The following section, which elaborates on the aspects of service-oriented metamorphosis, describes in detail the role of a solution service in the service-oriented modeling process.

SERVICE-ORIENTED MODELING DISCIPLINES DRIVE SERVICE METAMORPHOSIS. During the service-oriented modeling process, a service's state changes from intangible to physical. This transformation is driven by the service-oriented disciplines that we are commissioned to pursue. The normal course of evolution starts at a service's inception, during which a service emerges as a concept, continues through its analysis and design phases, and finally evolves as a solution service—a physical executable. By and large, this path is implemented with newly created services.

But can a physical solution service transform back into its previous intangible state? The necessity to iterate through service modeling cycles leads to the conclusion that a service, no matter how physical and established it is, will likely revert to an intangible state down the road. Subsequently, it will find its way back to the production environment to continue carrying out its operation. This path is not uncharacteristic of a software product. In fact, we often find that a service is "cloned" and simultaneously appears in two environments: in a production environment to ensure business or technological continuity, and in a design environment in which it is being examined for future enhancements.

Our service-oriented modeling practice advocates the transformation of a service from one state to another to maximize analysis, design, and architectural flexibility. This modeling elasticity often facilitates business agility, due to the service metamorphosis model. Exhibit 1.5 illustrates this concept. It depicts a service-oriented asset that undergoes four transformations:

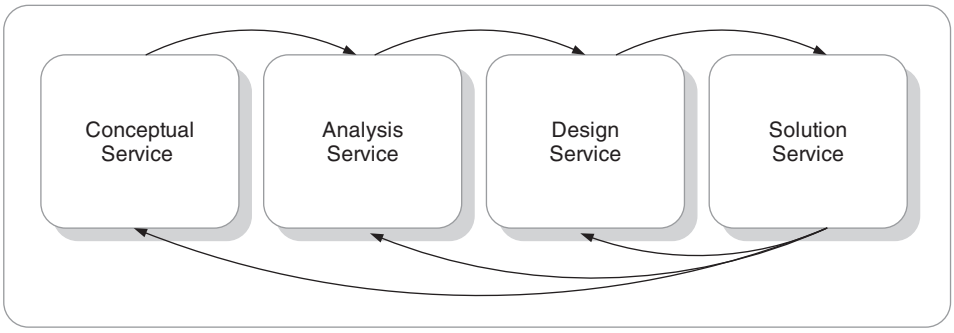

EXHIBIT 1.5 SERVICE-ORIENTED METAMORPHOSIS SCENARIOS

from a conceptual service to analysis, to design, and finally to solution service. Note that a solution service can also revert to the conceptual, analysis, and design service states.

From Solution Back to Conceptual Service. Under what circumstances should a solution service be transformed from its physical state back to an intangible formation—a conceptual service? The scenario, under which it may be required to treat a concrete service as an abstraction, is characteristically related to service redesign initiatives driven by changing business or technological requirements. Imagine a customer support service that must undergo functionality modification in light of a newly introduced electronic bill-payment feature. Or a fundamental change to a business model of an equity trading firm whose mission is to launch a new fixed-income line of products. Obviously, these reengineering efforts would require design and architecture teams to identify new concepts and expand on the existing ones. These efforts pertain to small- or large-scale projects.

Thus, the urge to reevaluate a concept on which a solution service is based is a substantial undertaking for any organization; a concrete service that must undergo a "surgery" of this nature is in essence set to restart its development life cycle. This reincarnation would also require reevaluation of the service's initial technology or an implementation that it is founded on.

Exhibit 1.6 depicts the four major reasons why a solution service may devolve into a conceptual service: modification to business or technological requirements, modification to business model and mission, concept reevaluation, and restructuring and reengineering.

From Solution Back to Analysis Service. Service reengineering initiatives do not always require the transformation of a solution service to its initial state—conceptual service. In many instances, the physical service's conceptual foundation may be sound enough and will not require altering the concepts it presents. However, if a solution service that executes a business or technological solution requires a functionality reevaluation, we should consider reverting it to its analysis state.

Why would an organization reexamine the performance inadequacies of a solution service? The chief reason is related to a particular service operation or another as well as to the collaborative efforts of services to provide a solution. For example, imagine the dissatisfaction

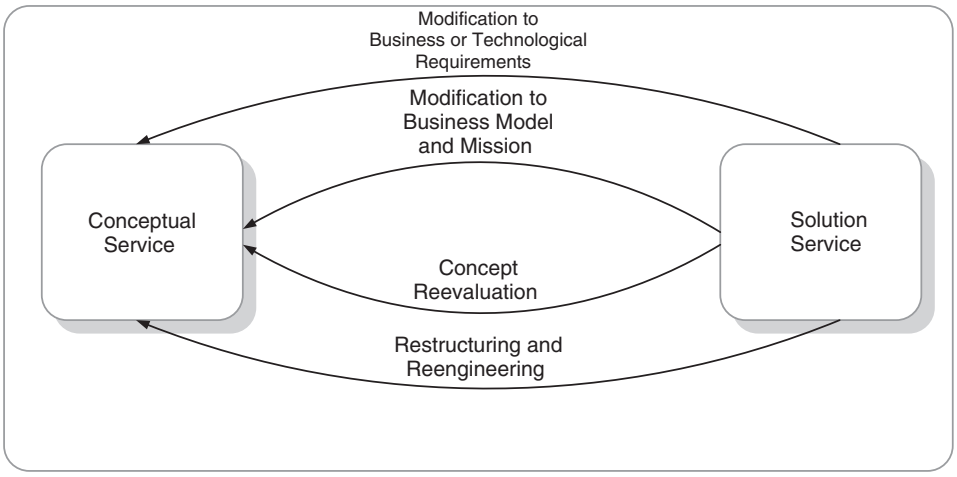

EXHIBIT 1.6 SOLUTION TO CONCEPT TRANSFORMATION

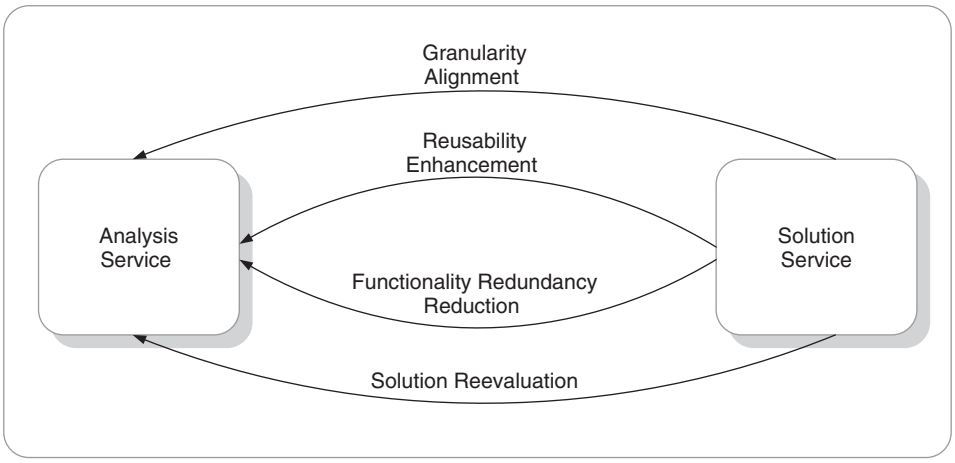

EXHIBIT 1.7 SOLUTION TO ANALYSIS SERVICE TRANSFORMATION

that can be caused to a banking institution consumer who receives a monthly statement that includes savings and checking account information but excludes trading account transactions. Larger-scale business initiatives can also spur solution reevaluation through pursuing analysis activities. These activities can be triggered by events such as a business expansion requiring additional functionality, application decommissioning, or even a merger or acquisition.

Other driving aspects behind the transformation of a solution service to an analysis asset are related to reusability enhancement, reduction of functionality redundancy, and even performance improvement. We can achieve these goals by reexamining a solution service's internal and external structures and validating its granularity. Granularity pertains to the level of business or technological functionality and the number of processes executed by a service. A coarse-grained service contains a large number of pursued activities. Conversely, a smaller-scale service that offers fewer processes is known as a fine-grained entity. Thus, reanalyzing service granularity would enable us to fine-tune its size and the scale of its functionality.

Exhibit 1.7 illustrates the four major reasons for the transformation of a solution service into an analysis entity: (1) granularity alignment, (2) reusability enhancement, (3) functionality redundancy reduction, and (4) solution reevaluation.

From Solution to Design Service. A solution service should be reverted back to its design state if a contract with its corresponding consumer is about to be modified. More specifically, this transformation should take place if requirements dictate modification to a service's collaboration, interaction, and integration with its peer services and subscribing consumers. This alteration of service behavior obviously can influence the coexistence conditions in a service community. "Behavior," for that matter, is associated with business or technical functionality activities managed and executed during transactions. This includes frequency of message delivery, coordination of message transmissions, reaction to message requests and responses, management of service availability, and even the method by which a service manages allowable consumption rates.

The following three major conditions suggest redesign of a solution service:

1. Changes in associations between services due to message exchange route alterations, the consolidation or decommission of a service that may affect the operating environment,

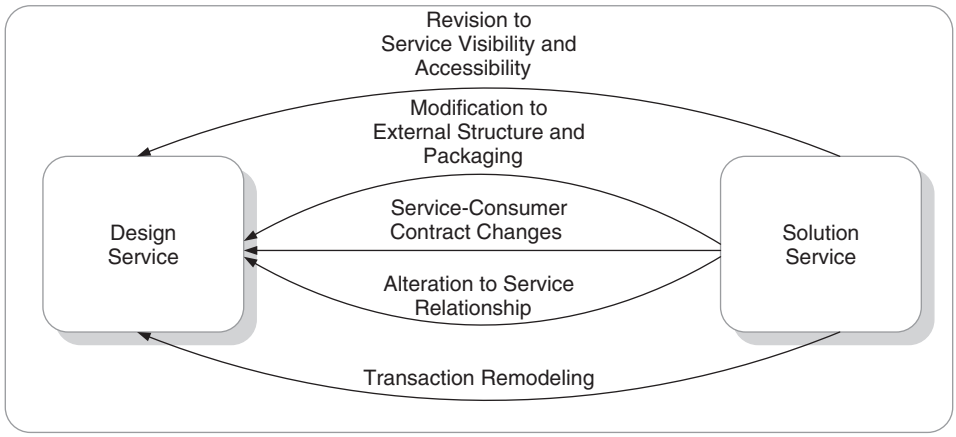

EXHIBIT 1.8 SOLUTION TO DESIGN TRANSFORMATION

and the addition of a service that delivers new functionality. These aspects may influence a service's accessibility and visibility in the environment in which it operates.

2. Changes that influence external service structures and a packaging solution that is deployed to production. External structures pertain to an overall logical composition of a solution in which services are arranged in certain formations (patterns) to achieve a design strategy, such as interoperability, reusability, asset consolidation, and loose coupling. This stylized arrangement of solution services is discussed in greater detail in Chapter 13.
3. Changes to business or technological functionality that triggers transaction redesign or enhancements, which characteristically would require reorchestration of transaction activities. Orchestration is typically affiliated with the overall workflow of message exchange between consumers and services. This includes coordination of message exchange operations and synchronization of message request and response activities.

Exhibit 1.8 illustrates the major conditions that lead to the redesign of solution service: revision to service visibility and accessibility, modification to external structure and packaging, contract changes, alteration to service relationship, and transaction remodeling.

SERVICE-ORIENTED MODELING DISCIPLINES: INTRODUCTION

A modeling discipline is a field of knowledge that offers best practices, standards, and policies to facilitate service-oriented development activities during a service's life cycle. We typically practice modeling disciplines prior to a service's construction and deployment to a production environment. We follow modeling discipline policies to avoid premature commitments to large budget allocations and lengthy service development periods. Modeling disciplines also enable us to analyze business and technological solution propositions formulated in a project inception phase to satisfy business and technological requirements.

Service-oriented modeling disciplines identify the core processes in which business and IT personnel must be engaged when producing design and architectural artifacts. These may include a variety of project deliverables, such as diagrams, charts, and documents. In addition, we are commissioned to prepare a working prototype, a software executable that implements some business or technological functionality regarded as a core challenge of the modeling process; this

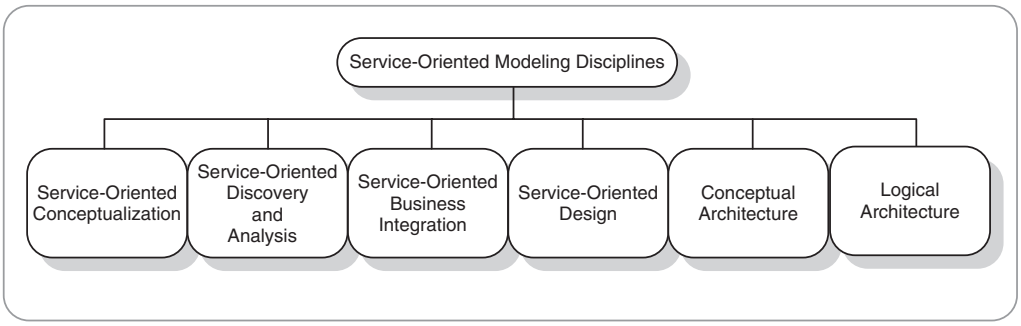

EXHIBIT 1.9 SERVICE-ORIENTED MODELING DISCIPLINES

is known as proof-of-concept. These deliverables will be handed to the service construction and deployment teams later on in the service-oriented development life cycle. (The service life cycle is described in detail in Chapters 2 and 3.)

What are these modeling disciplines that must be rigorously followed throughout the service development life cycle? Service-oriented modeling disciplines intrinsically focus on six areas of expertise, as depicted in Exhibit 1.9: (1) service-oriented conceptualization, (2) service-oriented discovery and analysis, (3) service-oriented business integration, (4) service-oriented design, (5) conceptual architecture, and (6) logical architecture. These disciplines are preliminarily presented in the sections that follow and are discussed in detail throughout the book (see Chapters 4 through 16).

SERVICE-ORIENTED CONCEPTUALIZATION. The service-oriented modeling process starts from the service conceptualization phase in which the driving concepts behind future solution services are identified. But here the focus is not on concrete software implementation. What is required is to establish a set of abstractions, a general terminology that offers solutions to business or technological requirements. It is important to remember that the conceptualization discipline advocates following a concept discovery methodology and process that ultimately yields intangible service-oriented assets known as *conceptual services*.

Exhibit 1.10 illustrates the two major milestones in the service conceptualization process: *attribution analysis* and *conceptual service identification*. (For more about the service-oriented conceptualization process, refer to Chapters 4 and 5, which provide a step-by-step activity guide and a set of deliverables recommended by this discipline.)

Attribution Analysis. To be able to efficiently ascertain concepts, it is necessary to scrupulously study business requirements, problem domain statements, and product specification documents to facilitate remedies to business or technological concerns. This inspection process requires that we identify the various attributes that a software product must possess. Hence, the *attribution analysis* process yields a set of core attributes that will be utilized to identify conceptual services.

Conceptual Services Identification. As a part of the conceptual services identification process it is necessary to

1. Ascertain conceptual services by utilizing the core attributes set that is established in the attribution analysis phase. This derivation process also facilitates the establishment

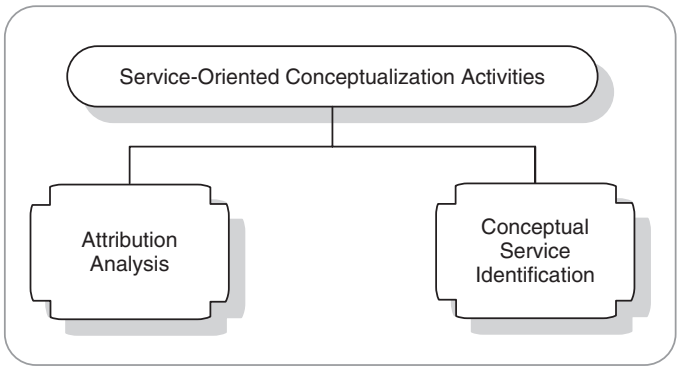

EXHIBIT 1.10 SERVICE-ORIENTED CONCEPTUAL MODELING PROCESS

of service taxonomies that can serve as organizational common language and glossary to enable future service categorization activities.

2. Find out how concepts are related to each other in terms of their business context and the process they embody. This concept association activity would enable us to understand service commonalities, analyze dissimilarities, and even foster reusability strategies early in the game.
3. Identify conceptual service internal and external structures. The most rudimentary formation is the *atomic* structure, which presents an indivisible conceptual service. The internal structure pertains to containment aspects of a conceptual service and its capacity to aggregate other concepts named *composite structure*. Conversely, the external service structure identifies formation of service groups known as the *service cluster*.

SERVICE-ORIENTED DISCOVERY AND ANALYSIS. The service discovery and analysis process is yet another discipline that enables us to continue identifying services that can contribute to a business or technological solution. We pursue this goal by analyzing the business and technological propositions presented thus far. More important, service-oriented analysis best practices advocate that we verify that the proposed service abstractions—the conceptual services devised in the conceptualization phase—do indeed provide a viable and valuable solution to the arising business or technological problems.

But the conceptual services that we have discovered so far may not provide a complete remedy to an organizational concern. Therefore, the general rule of thumb should be to continue to inspect existing legacy software assets and examine their capability to augment the solution that is being proposed. To accomplish this task, all service-oriented assets that participate in a solution are regarded as analysis services and engage in three service-oriented discovery and analysis activities, as depicted in Exhibit 1.11: service typing and profiling, service analysis, and service analysis modeling. Chapters 6, 7, and 8 discuss these processes in detail and elaborate on various discovery and analysis best practices.

Service Typing and Profiling. To be able to efficiently manage an organizational service portfolio and identify a service's potential contribution to a solution—either a conceptual or a legacy asset—we should be engaged in a service classification process. This service typing activity

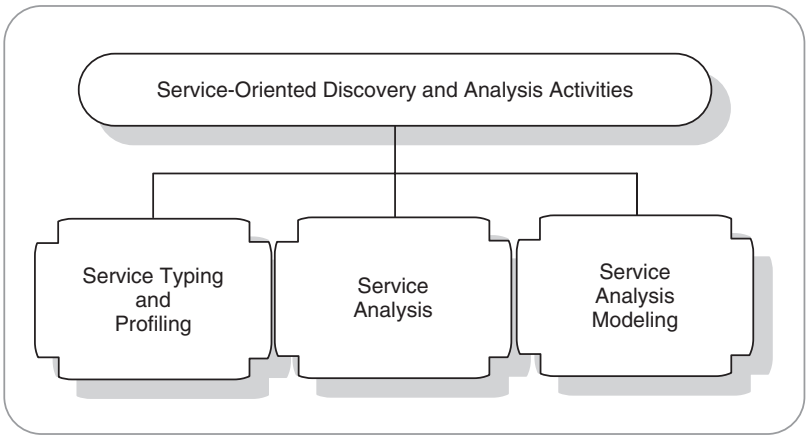

EXHIBIT 1.11 SERVICE-ORIENTED DISCOVERY AND ANALYSIS MODELING PROCESS

enables us to label a service based on its organizational identity and business or technological context, its internal structure, and its origin. The information that is collected by this process will facilitate the establishment of a service profile that can be utilized in future projects and business ventures.

Service Analysis. The service-oriented analysis process is the next activity that we need to pursue. Here we determine whether a service is viable to participate in the solution proposition. To accomplish this task we start with a service granularity assessment that reveals the business and technological functionality that each participating service contributes to the solution. We then continue with an analysis process that both validates the practicality of our services and explores service reusability, loose coupling, and asset consolidation opportunities.

Service Analysis Modeling. The analysis modeling activity concludes the service-oriented discovery and analysis process. The provided language is employed to facilitate a service analysis proposition diagram that captures aggregated and distributed service formations and identifies a collaborative service process scheme that proposes a solution to a problem that is being addressed. This modeling process fosters reuse of organizational software assets by utilizing enterprise legacy software and even organizational concepts represented by conceptual services.

SERVICE-ORIENTED BUSINESS INTEGRATION. A service-oriented business integration process is a discipline that should be practiced continuously throughout the service development life cycle. This is an ongoing activity that assures a close tie between the business and technology organizations. Most important, it fosters alignment between technological direction and an enterprise's business model and business strategy. But how can services be integrated with business imperatives? What does it mean to "align" an organizational technology strategy with business necessities? The answers to these questions have been studied by various institutions, and the conclusions were straightforward: To enable efficient integration between business and technology initiatives, enterprise business architecture practices must exist.

What is business architecture and how can it facilitate service-oriented business integration activities? Business architecture is simply organizational best practices and standards that describe what type of firm-wide architectures must exist for an enterprise to promote its business and

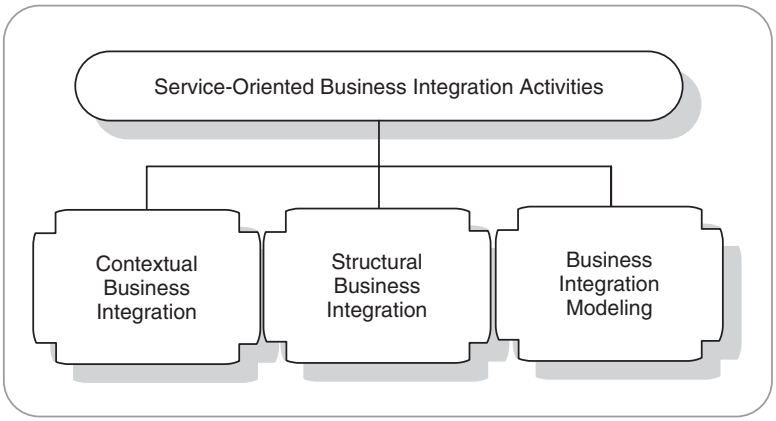

EXHIBIT 1.12 SERVICE-ORIENTED BUSINESS INTEGRATION MODELING PROCESS

increase revenue. It is "architecture about architectures." This is a business framework that not only depicts high-level business requirements but also identifies business functions, activities, and processes, along with business domains such as lines of business, geographic locations, and management control structure.

Exhibit 1.12 illustrates the three major activities that make up the service-oriented business integration process: contextual business integration, structural business integration, and business integration modeling. We regard all service-oriented assets that participate in a business integration venture as analysis services. Readers can learn more about these mechanisms and their best practices in Chapters 9, 10, and 11, which elaborate on service-oriented business integration process and modeling techniques.

Contextual and Structural Business Integration. The service-oriented business integration discipline advocates that we integrate our discovered services with two distinct views of our organizational business architecture: (1) a contextual perspective, in which we align our services with the enterprise business model and business strategies and (2) a structural perspective, which typically depicts the various business domains, such as lines of business and organizations, and business federation structures, such as geographic locations and management control.

Business Integration Modeling. Finally, the service-oriented business integration discipline offers an integration language that can facilitate various views of service alignment with business architecture perspectives. Here, the services that integrate with business domains are aligned with lines of business, business organizations, existing business products, enterprise business model, and business strategies.

SERVICE-ORIENTED DESIGN. The service-oriented design discipline is a logical perspective of the modeling process. It represents a wireframe version of the solution that is being proposed. "Wireframe" means simply connecting the dots by planning the message and information exchange flow between services and their corresponding consumers—this is typically dictated by the contracts that these involved parties committed to. The design modeling discipline also facilitates the establishment of logical relationship, interface, interaction, collaboration, and structural aspects of the participating consumers and services in the final design artifacts. Here, the focus is on the

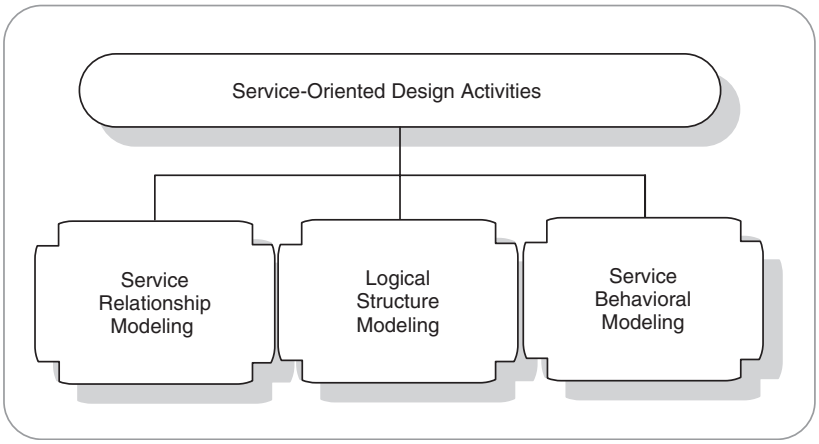

EXHIBIT 1.13 SERVICE-ORIENTED DESIGN MODELING DISCIPLINE PROCESS

capacity of services and consumers to comply with the SLA established prior to the modeling phase.

Exhibit 1.13 illustrates the three major service-oriented design modeling activities that establish the relationship, structure, and behavioral logical design aspects: *service relationship modeling*, *logical structure modeling*, and *service behavioral modeling*. These service-oriented design modeling aspects are discussed primarily in Chapters 12, 13, and 14. There readers will find the modeling implementation detail and best practices that will guide their service design phase.

Service Relationship Modeling. One of the most important aspects of the service-oriented design discipline is the various associations that a service maintains with its peer services and consumers. The major best practices and standards offered by the design modeling discipline address the following service relationship design concerns:

- How can we efficiently coordinate message and information exchange between consumers and services?
- How can we manage public exposure of services and yet grant safe access to subscribed consumers?
- Can we isolate services by limiting their visibility?
- How can we design an overall relationship view of our participating service-oriented assets?

Logical Structure Modeling. Remember, the relationship aspect of our service-oriented design, discussed previously, does not address an overall view of all the services and consumers that participate in a solution. Therefore, besides the service association discovery activity, it is also necessary to represent the solution that is being proposed from a structural perspective. Specifically, this effort pertains to building a logical formation, a design composition in which services are "glued" together to form a packaged structure that proposes a design modeling strategy. This approach pertains to service reusability, loose coupling, and solutions to interoperability challenges.

Service Behavioral Modeling. The service-oriented design discipline advocates that we also view a service and consumer interaction from a behavioral perspective; meaning the logical

design paradigm must also present business and technical functionality that services offer to their subscribed consumers. Therefore, the design modeling discipline advocates that such a perspective should be described by a transaction context. A transaction embodies service functionality, activities, message coordination, and interaction. These aspects are managed by message orchestration between service-oriented assets.

CONCEPTUAL AND LOGICAL ARCHITECTURE. What does it mean to architect a service-oriented environment? What are the major activities advocated by the service architecture modeling discipline? The service architecture process not only relies on conceptualization, analysis, business integration, and design artifacts; it also combines all the deliverables to create a service and consumer community. A service-oriented ecosystem is established in which services, consumers, and their empowering platforms can coexist, collaborate, and interface to provide viable business and technology solutions. Yes, we are engaged in an architectural endeavor that simulates a deployment environment, a production landscape that fosters our business model and strategies. We no longer address internal service structures, we merely tackle interaction and collaboration challenges between packaged solutions. This process offers two major architecture modeling disciplines, as depicted in Exhibit 1.14: the conceptual and the logical. For additional information, refer to Chapters 15 and 16, which provide in-depth service architecture modeling mechanisms and best practices.

Conceptual Architecture Modeling. The underlying physical construct of an architectural blueprint must be a sound technological environment that complies with architecture strategies, best practices, and standards. But even before the tangible aspects of a proposed architecture can be addressed, a strategy and a general direction must be devised that provide guidance for future technological implementation. The service-oriented conceptual architecture process is designed to undertake such tasks. This activity can assist in the discovery of the main *architecture concepts* that can drive future development and production projects.

The architecture abstraction discovery process can even start at the inception phase of a given project and continue throughout the development life cycle. Here, architecture abstractions are discovered that *describe* the technological environment. It is simply a matter of generalizing technology needs to facilitate an implementation plan for the future production landscape. These

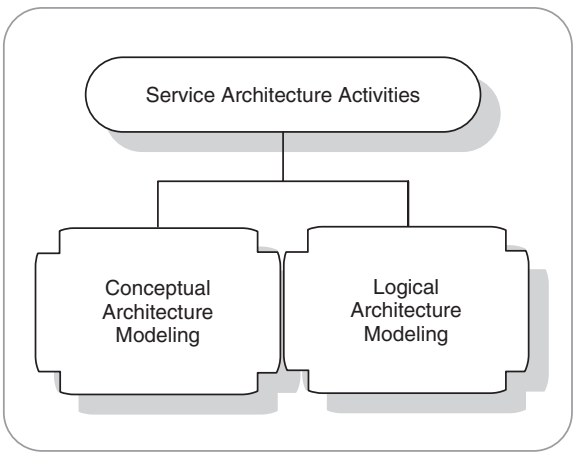

EXHIBIT 1.14 SERVICE ARCHITECTURE MODELING DISCIPLINE PROCESS

architecture imperatives may include service-oriented assets, such as third-party vendor products, existing services, middleware, and software platforms.

Logical Architecture Modeling. The other important driving aspect of the service-oriented architecture modeling discipline is a logical architecture perspective in which asset utilization, consumption, reusability, interoperability, and loose coupling best practices are being addressed. This process is chiefly concerned with the logical aspect of the production environment, in which service-oriented assets (services, legacy applications, language platforms, and middleware) must coexist and collaborate. Therefore, an asset utilization diagram provides detailed interaction between the deployed software packages and further elaborates on consumption and reusability requirements of a logical architecture.

MODELING ENVIRONMENTS

A modeling environment is not necessarily a physical location where the service-oriented modeling process takes place. In fact, it is a management framework that not only facilitates practicing service-oriented modeling disciplines, but also leads to recruiting personnel with the right expertise to manage the modeling process. To better understand a modeling environment, remember that each modeling discipline cannot be practiced in a vacuum. Surroundings are the major contributors to modeling activities. These may include modeling facilities such as software modeling tools, training aids, available documentation, and even a laboratory to test modeling assumptions.

Thus, a modeling environment is about the four Ps: people, planning, process, and policies. *People* denotes the personnel involved in guiding and enforcing modeling disciplines. These are typically the organizational center of excellence group or the SOA governance organization. In addition, business and technology personnel are also a part of the modeling efforts. The *planning* identifies the strategic or tactical aspects of the process and the required deliverables of the corresponding modeling discipline. These can include project plans, strategy documents, design and architectural blueprints, and diagrams. The *process* aspect is related to the sequence of activities that business and technology personnel pursue to achieve modeling goals. In addition, the *policies* pertain to how a solution can be proposed in the environmental framework. This reflects the management perspective, checks and balances, control of a modeling environment, and modeling discipline best practices and standards.

The service-oriented modeling paradigm supports three major modeling environments: *conceptual*, *analysis*, and *logical*, as depicted in Exhibit 1.15. Note that each of these frameworks

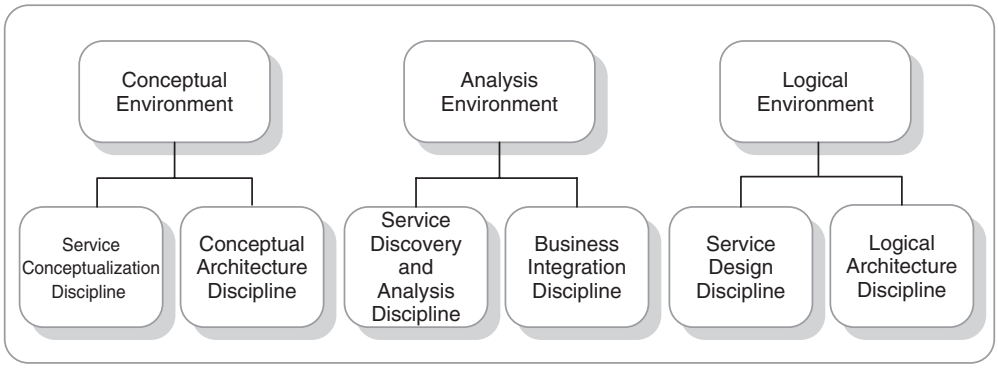

EXHIBIT 1.15 MODELING ENVIRONMENTS AND THEIR CORRESPONDING DISCIPLINES

also facilitates their corresponding modeling discipline activities. For example, the conceptual modeling environment provides a supporting framework for the discovery of conceptual services when one pursues the service conceptualization discipline. In the same fashion, the analysis environment facilitates all service discovery and analysis and business integration discipline tasks.

CONCEPTUAL ENVIRONMENT. An established methodology that can facilitate the formalization of undocumented ideas and propose solutions to enterprise concerns is a necessity for most organizations. A conceptual environment provides a management framework that promotes proper analysis of the problem domain and business and technical requirements and also assists with documenting solution propositions. This framework intrinsically drives two major service-oriented modeling initiatives that are pursued at different times during the service life cycle: constituting enterprise conceptual services during the service-oriented conceptualization phase, and establishing architectural concepts devised during the conceptual architecture process. The major activities that a conceptual environment can assist with include:

- Facilitating the studies of the organizational business model, business strategies, and problem domain. This effort includes inspecting the state of business, business analysis, and business activities.
- Involving business and technology personnel in whiteboard and conceptualization sessions to formalize ideas and establish organizational concepts. This may include generalization of problems, abstraction identification, and consolidation of ideas that emerge from various organizational sources, such as people, business processes, and existing documentation.
- Helping business and service-oriented architects, modelers, and developers to establish enterprise conceptual services.
- Facilitating the conceptual architecture process, in which technological abstractions are identified and categorized.
- Enabling the constitution of an organizational service-oriented taxonomy consisting of conceptual services and architectural concepts and managed by an asset portfolio manager software package.

ANALYSIS ENVIRONMENT. The analysis environment facilitates the transformation of a service from its initial conceptual state to an analysis service for further inspection and categorization. This process is pursued to encourage discovery and profiling activities of services that may participate in a service-oriented solution. The analysis environment framework also provides a platform for analysis modeling, allowing architects, asset portfolio managers, business architects, business analysts, and developers to conduct sessions that yield an analysis proposition model.

In addition, the analysis environment management framework supports analysis activities throughout the service-oriented business integration phase, during which business integration modeling disciplines are applied. This alignment activity requires the collaboration of business and IT personnel to identify proper service integration opportunities and to establish a business integration model that depicts the alignment scheme between services and business domains.

Consider the following analysis environment facilitation activities:

- Fostering research initiatives that identify analysis modeling and visualization tools.
- Assisting with asset portfolio analysis to identify service-oriented asset candidates that can participate in a solution. This activity typically involves portfolio managers, product managers, business analysts, and architects.

- Coordinating analysis sessions to enable analysis operations such as service decomposition, aggregation, and unification.
- Facilitating analysis modeling sessions to devise an analysis proposition. These activities yield a service-oriented analysis model that can be further utilized in the design and architecture phases.
- Encouraging alignment of business and technology strategies that can facilitate proper service-oriented business integration.

LOGICAL ENVIRONMENT. A logical environment supports the transformation of analysis services into more tangible assets called design services. This environment also depicts a visual view of the logical landscape in which service relationship, structures, and behaviors illustrate a wireframe solution to an organizational concern. Moreover, on the service architecture front, the logical environment framework facilitates the creation of a logical architecture that depicts an integrated service-oriented ecosystem where packaged services collaborate to provide a viable solution. This includes the following activities:

- Conducting service-oriented design and architecture training sessions to promote best practices and standards
- Encouraging research integrated development environments (IDEs) that offer service relationship establishment capabilities, service structure modeling, and transaction modeling
- Utilizing software tools to enable efficient message orchestration and expression of logic flow
- Providing laboratory facilities so that a proof-of-concept can be conducted to examine the proposed service ecosystem
- Facilitating the establishment of contracts between consumers and their corresponding services

SERVICE-ORIENTED MODELING FRAMEWORK

Before moving on to the chapters that describe in detail the modeling disciplines and their activities, let us take an overall view of the service-oriented modeling framework. This work structure is chiefly a high-level map depicting the various components that contribute to a successful service-oriented modeling approach. It illustrates the major elements that identify the "what to do" aspects of a service development scheme. These are the modeling pillars that will enable a practitioner to craft an effective project plan and to identify the milestones of a service-oriented initiative—either a small- or large-scale business or a technological venture.

Exhibit 1.16 depicts the four sections of the modeling framework that identify the general direction and the corresponding units of work that make up a service-oriented modeling strategy: *practices*, *environments*, *disciplines*, and *artifacts*. Remember, these elements uncover the context of a modeling occupation and do not necessarily describe the process or the sequence of activities needed to fulfill modeling goals. These should be ironed out during the project plan that typically sets initiative boundaries, timeframe, responsibilities and accountabilities, and achievable project milestones.

MODELING PRACTICES. A framework practice describes an occupation, a major function that should be pursued to drive modeling activities. The practitioners, for that matter, are the business and technology personnel engaged in analysis, design, and architectural aspects of services. They also offer vital input into the management process of the modeling efforts. A collaborative approach to furnishing solutions in the business and technology–specific area of expertise would yield successful modeling artifacts. The typical participants who address the concerns of

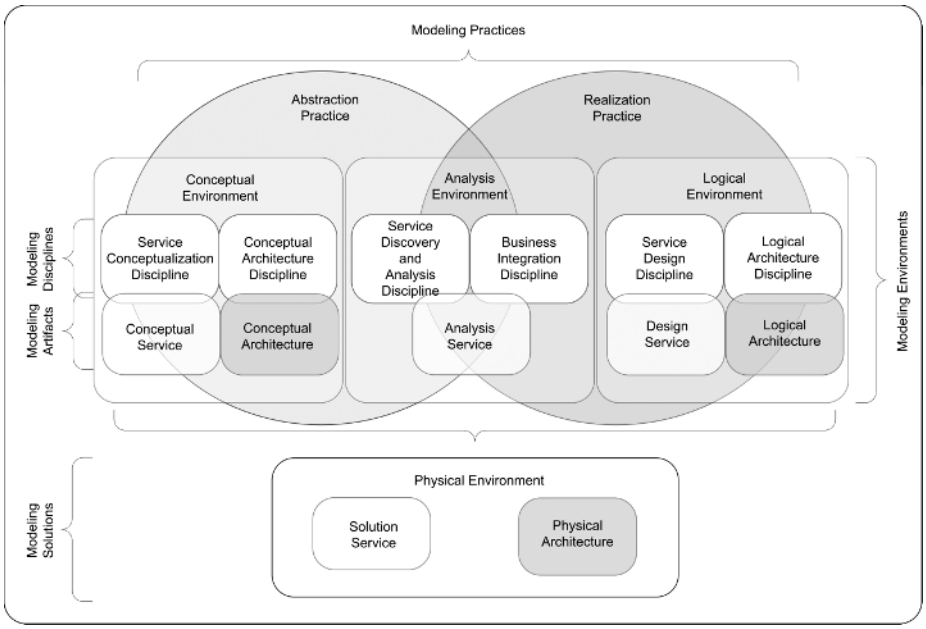

EXHIBIT 1.16 SERVICE-ORIENTED MODELING FRAMEWORK

a framework practice are business and technology managers, service-oriented strategists, asset portfolio managers, business analysts, business architects, technical architects, developers, and, of course, software modelers, such as service and database modelers.

This service-oriented modeling framework identifies two major modeling practices that drive and influence all modeling activities during a project or any business initiatives: abstraction and realization.

ABSTRACTION PRACTICE. What are the major modeling concerns addressed by the abstraction practice? Why is such a practice vital to any modeling effort? Every software development initiative will reveal new ideas and establish the driving concepts that the project and its deliverables will be founded on. Remember, the major concerns of the abstraction practice, on the one hand, revolve around the discovery aspects of services, and on the other, depict an architectural environment based on abstractions. These two major activities may not take place in any sequence, but they are well embedded in the abstraction practice activities category. The major questions that are typically asked by those engaging in the abstraction practice are

- What are the major business concerns that drive this initiative?
- What do the business or technological requirements recommend?
- What services do we actually build?
- What assets do we reengineer?
- What kind of metaphors do we use to describe our services?
- How do we conceptualize our architectural environment?
- How do we identify our service architecture needs and landscape?
- Which personnel are involved?

Separating Concerns and Generalizing Problems. The service-oriented abstraction practice enables practitioners to generalize private instances of organizational problems and ignore, for the time being, implementation details of the current services and the supporting technological environment. They will be able to break down organizational concerns into more manageable abstracted areas of interests. This process is known as separation of concerns. Practitioners will also be encouraged to inspect the problem domain and understand the "big picture" scenario by analyzing the business imperative, the business model of their organization, and the strategies designed to generate revenue for the organization.

Employing a Horizontal Modeling View. The abstraction practice, also known as horizontal practice, typically engages the cross-organization stakeholders that represent various silo operations in the enterprise to collaborate on service-oriented modeling activities. Imagine how beneficial it would be to involve representatives from different lines of business to discuss how they can collaborate to foster asset reusability opportunities and to encourage reduction of functionality redundancy and consolidation of future assets. They should also be able to identify mutual organizational concerns and propose remedies that can be reused across the enterprise. Imagine how advantageous it would be if these business and technological practitioners partner to meet service interoperability challenges and to enable a holistic view of all modeling assets regardless of their origin, geographic location, or empowering technologies.

Visualizing Abstraction Practice Process and Artifacts. Can the abstraction practice enable one to visualize the activities and deliverables involved? Can concepts be visible? Indeed, concepts are invisible entities. They originate from peoples' ideas and propositions. But the abstraction process should be mechanical and well formulated, driven by a well-defined process and should employ tools to enable visualization of intangible software assets. By "mechanical," we mean that the abstraction process employs conceptualization best practices that yield conceptual services, which, despite their intangibility, are treated like the other valuable assets in an organization.

Employing Corresponding Service-Oriented Modeling Disciplines. To ensure the success of the service-oriented modeling process, modeling practices must be guided and enforced by modeling disciplines. They were previously introduced in the Service-Oriented Modeling Disciplines: Introduction section, and they are discussed in further detail in Chapters 4 through 16. But these disciplines cannot operate in a vacuum. They require one to create the proper modeling environment. What are the corresponding disciplines of the abstraction practice? Take a second look at Exhibit 1.16. The depicted abstraction practice spans two modeling environments:

- **Conceptual environment.** As previously discussed in the section on service-oriented modeling environments, this supporting landscape facilitates two distinct disciplines that are chiefly concerned with the service-oriented conceptualization process. These are the best practices and standards that assist with the "how to do" aspects of a conceptual environment: First, the service-oriented conceptualization discipline facilitates the identification of conceptual services and establishment of organizational taxonomies. Second, the conceptual architecture discipline provides guidance for the discovery of technological abstractions that describe architectural imperatives and service-oriented assets, such as services, middleware, and platforms.
- **Analysis environment.** Remember, some activities that are pursued in the analysis environment are also regarded as a part of the abstraction practice, as is noted in Exhibit 1.16. The abstraction practice advocates that we pursue conceptualization activities and that we also identify, type, profile, and evaluate services by complying with service-oriented discovery and analysis disciplines, which are facilitated by the analysis environment. Furthermore, some of the service-oriented business integration discipline activities also fall

within the abstraction practice domain. Here we are commissioned to align our services with business strategies and structures such as geographic locations and even management structure.

REALIZATION PRACTICE. The realization practice typically starts when the abstraction practice runs its course, during which most ideas have been formalized and established as organizational concepts. These abstractions then transform into more tangible software entities that are regarded as logical units of analysis. The realization practice addresses these tangibility aspects that are articulated by the logical design of services, such as message exchange patterns, business or technological functionality, service structure, and the overall workflow and process orchestration. The questions that are typically asked during the realization process are:

- What is the process of transforming an analysis service to a design service?
- What does a logical design process entail?; What is a logical architecture?
- Who is involved?

The realization practice obviously overlaps with the abstraction practice because of some process commonalities that are typically pursued in both domains. This pertains to activities that are managed during the service-oriented discovery and analysis phase and business integration.

Creating a Virtual World. The ultimate goal of the realization practice is to transform intangible entities into more concrete software assets. However, this process should stop short of the construction of physical services. This practice is not about source-code building and deployment to production. Its main charter is to depict a virtual world of software assets that simulates a prospective technological solution that may be materialized by the end of the service life cycle. This is a virtual world in which services not only interact and collaborate to achieve a defined goal but also provide a supporting environment that empowers the business and technological functionality.

Centering on the Solutions. Modeling shortsightedness can only hamper the efforts to find a conclusive resolution to a business or technological concern. But along with recognizing what the large-scale problems are, we must first pinpoint the burning issues facing an organization. Furthermore, to be able to focus on a particular solution, business or technological requirements should be the driving motivation behind any modeling activity. In particular, these imperatives should guide modeling practitioners to tackle vital aspects of a solution and avoid propositions to a wide range area of concerns.

Employing Corresponding Service-Oriented Modeling Disciplines. The realization process supports two major modeling environments: analysis and design. As discussed previously, some analysis activities are pursued during the abstraction process, but they are also common to the realization practice. Therefore, the modeling analysis environment and its corresponding disciplines are regarded as shared modeling aspects for both modeling practices. The abstraction practice accentuates the discovery and establishment of services, while the realization practice yields design services that resemble a more concrete world.

- **Analysis environment.** The analysis environment facilitates analysis activities by employing the service-oriented discovery and analysis discipline. The realization practice, however, focuses more on the analysis modeling part of this discipline, in which an analysis proposition is crafted. (Again, the discovery activities are more of a concern to the abstraction practice that was discussed previously.) Moreover, the analysis environment enables the service-oriented business integration activities in which services are

aligned with business imperatives. Here the realization practice is concerned more with the business integration-modeling portion and less with the discovery of business alignment perspectives.

- **Logical environment.** The logical environment facilitates two disciplines that contribute to the design and architecture logical foundation: service-oriented design and logical architecture. These disciplines contribute the most to the formation of a virtual world that mimics the service ecosystem that we are about to construct.

PHYSICAL ENVIRONMENT. Service-oriented modeling activities can always be repeated to perfect the resulting artifacts that the modeling disciplines require for delivery. This perpetual process is called *iteration*. Iterating through the various disciplines should promote one major modeling agenda: to transform services from their intangible state to concrete entities. There are two major physical artifacts that make up a physical environment:

1. Deployable service-oriented software assets that should be integrated with a production environment. (These tangible entities are called solution services.)
2. A physical architecture that emerges from our logical architecture. This concrete architecture depicts an integrated and configurable production environment that typically consists of infrastructure, networks, management facilities such as monitoring and security, and, of course, solution services.

SUMMARY

The service-oriented modeling driving principles are virtualization, metamorphosis, and literate modeling.

Enterprise concepts, foundation software (such as middleware and language platforms), legacy software (such as applications and services), repositories, and utility software—are all conceived as organizational service-oriented software assets.

The service-oriented modeling process stakeholders are business and technology personnel that equally share the burden of software development in the organization. They also present two service-oriented modeling perspectives: business and technological views.

The service-oriented paradigm recognizes three distinct modeling services: conceptual, analysis, and design. They contribute to the establishment of a physical solution service.

There are six modeling disciples that drive service development efforts: conceptualization, discovery and analysis, business integration, design, conceptual architecture, and logical architecture.

The service-oriented paradigm supports three modeling environments: conceptual, analysis, and logical.

The service-oriented modeling framework identifies the "what to do" aspects of a service development environment.

Endnotes

1. *Journal of Digital Information*, Vol. 5, Issue 1, Article No. 298, July 16, 2004.
2. *www.en.wikipedia.org/wiki/Virtuality*
3. Edsger W. Dijkstra, "On the Cruelty of Really Teaching Computer Science", December 1988, p. 16. (*http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF*)