



Decomposing Composition: Service-Oriented Software Engineers

M. Brian Blake, *Georgetown University*

Vol. 24, No. 6
November/December 2007

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.



© 2007 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

For more information, please see www.ieee.org/web/publications/rights/index.html.

Decomposing Composition: Service-Oriented Software Engineers

M. Brian Blake, *Georgetown University*

How can we leverage traditional state-of-the-art software development life cycles to support development in service-centric software systems?

The explosion of information technology (including service-oriented architecture) and its underlying capabilities has led to the evolution of software development life cycles over the past three decades. Software engineers are continuously exploring approaches to software and system development that are domain, application, and technology independent. Early approaches included waterfall life cycles that promote creating concrete requirements before any significant design or development

occurs. However, this can lead to the misalignment of the system with the required business needs when requirements change or evolve.

This is an even larger concern in service-centric software systems. SCSSs promote an open environment in which Web services are universally available across organizations leveraging various supporting technologies. Because systems are constantly creating and removing services, not only are the requirements constantly changing but also the system (as defined by its underlying services). This results in an even larger separation of business needs from system capabilities.

To address this issue, software developers have adopted more incremental and iterative life cycles that let the engineers revisit requirements and design artifacts throughout development. Many life cycles also include agile approaches that encourage constant interaction between software engineers and their corre-

sponding subject matter experts (SMEs) in the application domain. These changes are a first step toward supporting development in SCSS environments. However, even more critical is the need for software development life cycles to become as modular as the service-centric environments they construct. On the basis of experience from my research group and industry consultancy, I propose creating a *service-centric system-management* life cycle that adapts more traditional software development phases to better address the issues that arise in dynamic SCSS environments.

A separation of concerns

Different stakeholders in the development process must assume different roles. As such, the development process can benefit from a separation of concerns that acknowledges the difference between the two key activities software engineers perform in SCSS development:

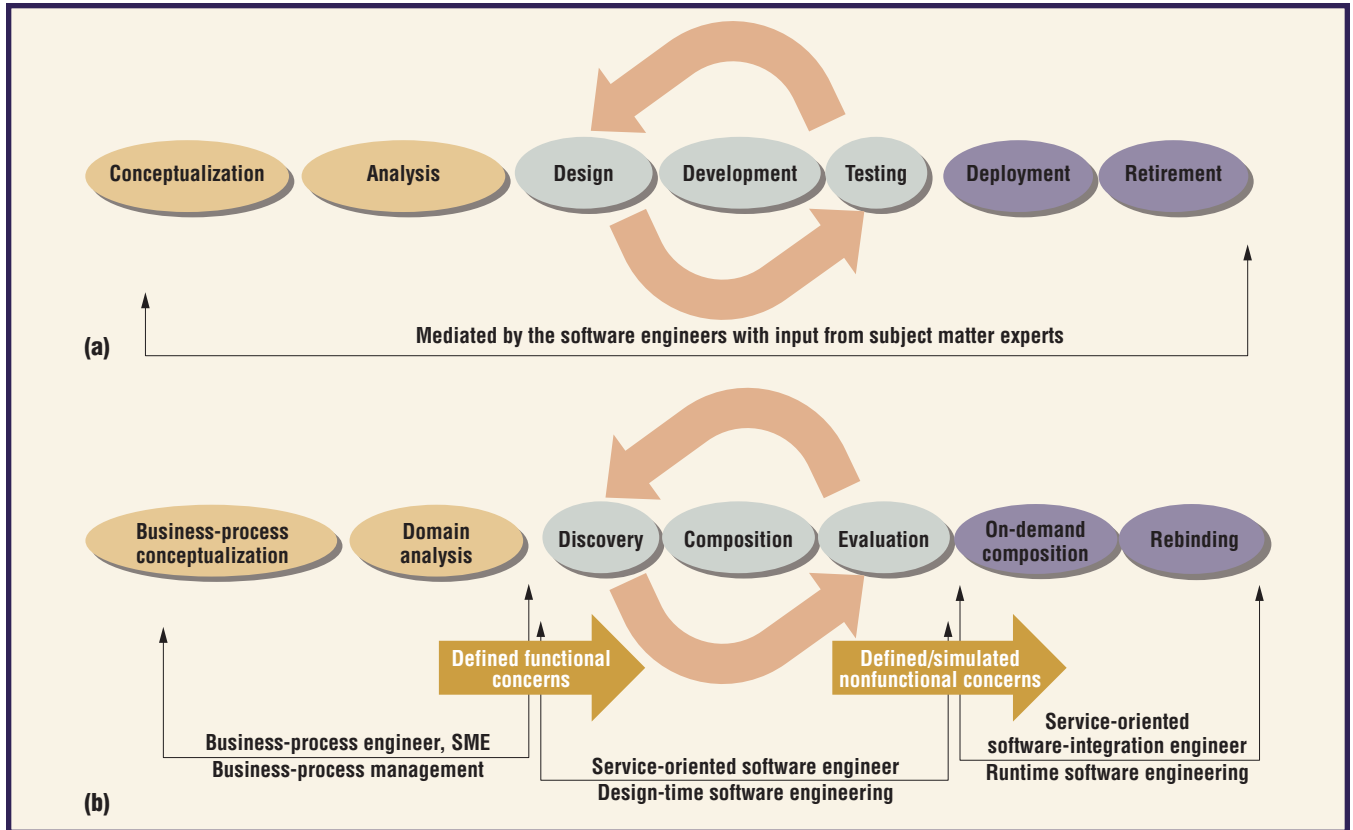


Figure 1. The software engineering life cycles for (a) service development and (b) service-centric software systems management.

service development and service-centric system management. This separation lets experienced individuals manage the software artifact most relevant to them.

Developing services

Service development follows the traditional notion of software engineering. Because Web services are the backbone of SCSS environments, software engineers must be skilled in developing modular software services that decompose business processes into bite-size pieces (that is, manageable subcapabilities). By decomposing capabilities into modular services, organizations can share their offerings at multiple levels of granularity while creating unique access points for their peer organizations.

Although software engineers must intimately understand business-process details, the life cycle for developing individual Web services will resemble traditional iterative processes such as the Rational Unified Process.¹ The seven general phases in this life cycle are conceptualization, analysis, design, development, testing, deployment, and retirement (see figure 1a).

During conceptualization and analysis, software engineers elicit requirements that clarify

the business needs. Design, development, and testing is an iterative set of phases in which the software engineers design, create, and evaluate services. They use test results as feedback to the design phase and then deploy the services for universal access. At some point, when the business offering changes, they remove and retire the services. Software engineers generally manage this life cycle from beginning to end, although they engage SMEs, business-process engineers, and application domain experts throughout.

Managing services

After developing and deploying the Web services, software engineers must engage in service-centric system-management life cycles to allow on-demand services to be discovered, analyzed, composed, and consumed. This environment is far more dynamic than the traditional operations associated with component or library-oriented software systems. Because service-based capabilities are managed in distributed locations outside the consumers' control, this environment requires more specialized development life cycles for maintaining predictable operations. Furthermore, service-oriented software engineering professionals

Related Work in Developing SCSSs

Developing effective service-centric software systems requires that system artifacts are refined by specialized engineers participating in the system's business process, design, and integration. Wei-Tek Tsai and his colleagues discuss static service-oriented architecture and dynamic SOA.¹ Although useful in certain settings, static SOA neglects the advanced notions of SOA (that is, on-demand discovery and composition) by addressing preselected, prearranged services. You can generally realize static SOA using traditional software engineering processes, which have historically worked well for deploying and incorporating software components or libraries.

My research lab's work advocates that dynamic SOA represents a domain with a richer set of problems. These problems are associated with the fact that consumers of services, in many cases, have no control over the volatility or responsiveness of services external to their own enterprise systems. Tsai and his colleagues identify specific business-process aspects important to SCSSs and map them to process models. Their approach leverages the notion of a Model-Driven Architecture;² they suggest that the resulting service-based models can be analyzed before development and deployment. Our approach extends their work by explicitly highlighting functional and nonfunctional concerns inline with the model. Our approach further facilitates the empirical analysis of the models via simulation and the stronger assessment of design decisions particularly when there are many design choices with just slight variations.

John Grundy and his colleagues also highlight nonfunctional concerns within a service-oriented systems context.³ Our work furthers their approach by suggesting that some nonfunctional concerns will be local (that is, known) and others will be external (that is, estimated and simulated). Our modeling approach lets engineers specify both known and estimated metrics in the design, before simulation.

Traditional software engineering methodologies set the foundation for software life cycles in these environments, but predictability requires adopting modular software life cycles and refining them into industry-strength, principled software

processes. Although UML is a widely accepted language for modeling, it lacks the ability to highlight functional models with nonfunctional specifications. Aside from providing general guidelines,⁴ few principled software engineering life cycles deal with the development and management of SCSSs.

Kunal Mittal specifies a service-oriented unified process (www.kunalmittal.com/html/soup.shtml). This is a promising intersection, but Mittal focuses on providing service-oriented characteristics within the existing Rational Unified Process.⁵ Our work defines new phases and a specialized iteration cycle that responds to the dynamic nature of service-centric software environments. In addition, our work identifies the distinct roles that separate the concerns of business process, software design, and system management.

David Cox and Heather Kreger also designate an SOA-solution life cycle that separates the "planes" of business processes, services, and infrastructure.⁶ These planes are similar to our phased life cycle. Our work extends these planes into two dimensions (engineering roles both for service development and for service management). Leveraging nonfunctional specification within a generic SLA development process is a further distinction.

References

1. W.T. Tsai et al., "Process Specification and Modeling Language for Service-Oriented Software Development," *Proc. Workshop Future Trends of Distributed Computing Systems*, IEEE CS Press, 2007, pp. 181–188.
2. J. Mukerji and J. Miller, *MDA Guide Version 1.0.1*, tech. report omg/2003-06-01, Object Management Group, 2003; www.omg.org/docs/omg/03-06-01.pdf.
3. J.C. Grundy et al., "Performance Engineering of Service Compositions," *Proc. 2006 Int'l Workshop Service-Oriented Software Eng. (Iwso6)*, ACM Press, 2006, pp. 26–32.
4. N. Gold et al., "Understanding Service-Oriented Software," *IEEE Software*, vol. 21, no. 2, 2004, pp. 71–77.
5. P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley, 1998.
6. D.E. Cox and H. Kreger, "Management of the Service-Oriented Architecture Lifecycle," *IBM Systems J.*, vol. 44, no. 4, 2005; www.research.ibm.com/journal/sj/444/cox.html.

must intervene to establish agreements between consumers and producers.

This new life cycle focuses on managing available services in real time, so the seven phases are slightly different: business-process conceptualization, domain analysis, discovery, composition, evaluation, on-demand composition, and rebinding (see figure 1b). Although business-process conceptualization and domain analysis also attempt to capture business needs, it's important to acknowledge that the solution services already exist. Discovery, composition, and evaluation

represent the core service-oriented computing paradigm² of finding, blueprinting, and analyzing candidate services at design time. Finally, at runtime, on-demand composition and rebinding construct and evolve composite service-oriented business processes over time.

Service-centric system management

The service-centric system-management life cycle is modular. You can split it into three aggregate phases: business-process management,

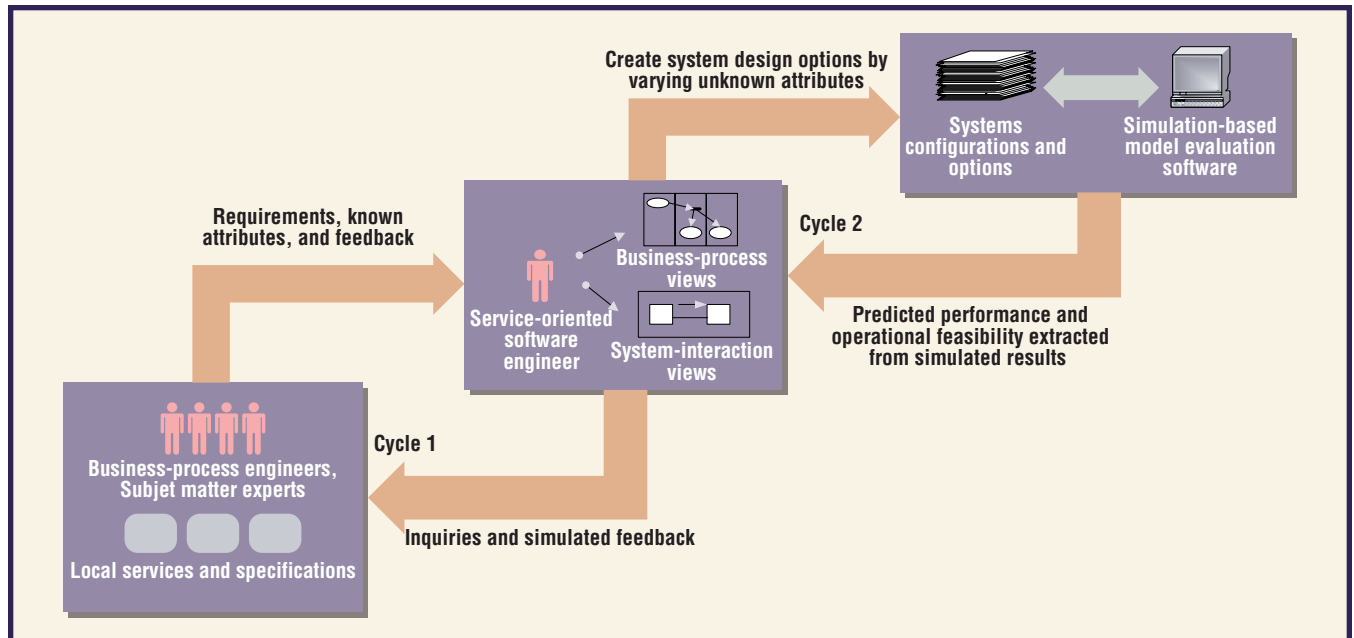


Figure 2. An iterative two-cycle process for engineering SCSSs at design time.

design-time software engineering, and runtime software engineering, and different stakeholders can manage these aggregate phases. (See the “Related Work in Developing SCSSs” sidebar.)

Phase 1: Business-process management

During the first phase, because services already exist with open specifications, business-process engineers and SMEs can evaluate existing services to see if they meet their business needs (see figure 1b). However, software engineers should also be involved so that they can begin to understand the application domain and can temper expectations by contributing their software design experience.

Phase 2: Service-oriented software engineers

Service-oriented software engineers must create long-standing, composable capabilities while integrating business-process knowledge with traditional software engineering methodologies. Considering the dynamic nature of future SCSSs, model-based software engineering over an iterative design and evaluation life cycle is critical.

An iterative design life cycle. Service-oriented software engineers analyze requirements for building new capabilities while concurrently analyzing whether a composed set of interorganizational services can fulfill those requirements. Two interleaving cycles thus exist (see figure 2).

In the first cycle, the engineer elicits require-

ments from business-process engineers and SMEs to determine their specific needs. At the same time, he or she collects technical information about local known services. Next, the engineer develops a business-process view—that is, a high-level, workflow-oriented view defining the process consisting of interorganizational Web services. He or she also develops a system-interaction view—a low-level view describing complex implementation protocols. The engineer models these views while incorporating feedback from the business-process engineers and SMEs.

In the second cycle, once the service-oriented software engineer feels that the views represent his or her understanding, he or she augments them with estimated information about services within the interorganizational boundaries. Integrating unknown external conditions with known local factors lets the service-oriented software engineer generate many optional designs and system configuration alternatives.

There’s a growing desire for automated software engineering approaches such as specialized model evaluation and simulation software that could assist with this integration.^{3,4} Such approaches could simulate models and present metrics about the predicted system operations to the software engineer. In related work,³ the COACHES (Coordination of Agents for Composing Heterogeneous Electronic Services) simulation tool automatically evaluates models (such as the software engineering models in this ar-

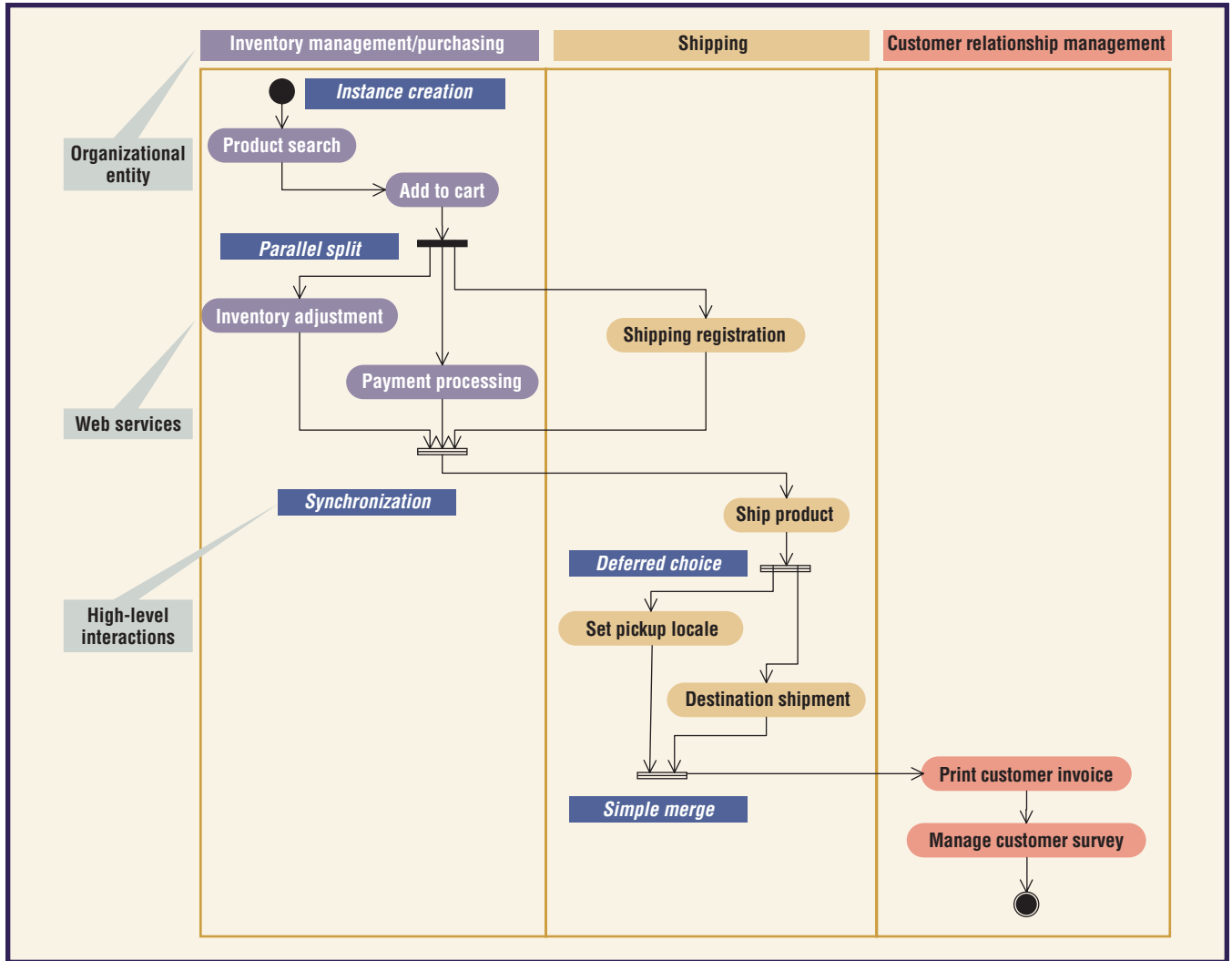


Figure 3. The business-process view.

ticle) and assists engineers in service-centric development strategies.

A two-level modeling approach. A modeling approach with multiple views at various hierarchical levels allows for incremental additions and enhancements as the system evolves. The two-level modeling approach of business-process views and system-interaction views leverages UML activity diagrams and communication diagrams, respectively.

The business-process view shows the sequence of services that perform a scenario for selecting and purchasing a product, arranging for shipment, and acquiring invoice and customer-satisfaction information (see figure 3). Software engineers annotate organization entities within UML swimlanes, and they model Web services using UML activities. They also annotate system interactions in the UML ac-

tivity diagram as associated with UML control flow notations, such as fork/join relations and initial/end state notations.

Similar to design patterns for object-oriented systems, you can leverage well-defined business-process patterns (workflow⁵ and service-interaction [www.serviceinteraction.com] patterns) of system interactions that generally occur in these types of environments. Figure 3 shows workflow patterns (such as Synchronization, Deferred Choice, and Simple Merge) as stereotypes that highlight interactions between Web services. While the business-process view allows collaboration between software engineers and SMEs about capabilities, lower-level system-interaction views (such as UML communication diagrams) capture the SCSS's technical operation.

System-interaction views show more detail of system actions by showing the system's step-

wise operations. Each step can be highlighted with performance-based information. With respect to performance modeling, I build on the UML Profile of Schedulability, Performance, and Time (see www.omg.org/technology/documents/formal/schedulability.htm). UPSPT is an existing framework for annotating scheduling, performance, and time information on functional models. For our approach, the performance modeling profile is most relevant. Figure 4a shows a recreation of the UPSPT with the introduction of additional SCSS-based extensions.

The UPSPT model is closely related to the operation modes in the SCSS environment. The `Workload` concept models system load. A `Closed Workload` represents a closed population of actors in the system. In a `Closed Workload`, an actor accesses the system, then experiences a delay before accessing the system again.

Our approach more closely relates to an `Open Workload`, where an open number of stakeholders have variable load distributions (`occurrencePatterns`). Although UPSPT includes multiple attributes, four attributes are relevant to our approach: the number of services, the number of agents, the number of changes to service bindings, and the reliability of the network and system components.

`PScenario` is similar to the concept of a business-process schema, where each `PStep` maps to underlying Web services. I extend this model by adding three service-oriented attributes: proximity, reliability, and connection speed. Each attribute is a numerical measure describing a particular Web service.

Finally, `PResource` (further specified as `PProcessingResource` and `PPassiveResource`) realizes SCSSs that represent organization entities. `PPassiveResources` represent system components that mainly execute in parallel to application-specific functions; however, their existence affects the system as a whole. Currently, our modeling approach focuses on `PProcessingResources`. I extend `PProcessingResources` (such as mainline functional capabilities) by adding three attributes: execution time, data management time, and communication time.

Figure 4b shows an example of the Instance Creation pattern, as shown in the business process view of figure 3, annotated with performance information. This example demonstrates how UPSPT can incorporate performance values, which are captured with the UML note

notation. The UML note is annotated with a stereotype that represents the associated object as defined in the UPSPT metamodel (see figure 4a).

The SCSS task execution is also annotated with execution time specifications. Again using an estimated mean, the system's execution time varies with the range of the queue size. For example, the execution time is 1 second when the queue contains between 0 and 10 requests; it's 4 seconds when the queue is between 40 and 2,000 requests. Three parameters specify all the performance values in UPSPT: the value's source (for example, required, assumed, predicted, measured, or estimated), the type of value (average, sigma, *k*th moment, or maximum), and the actual value with units.

Evaluating system performance. Once the SCSS environment is modeled with business-process and system-interaction views, the service-oriented software engineer can essentially decompose the composition routines. Web service performance can vary on the basis of multiple offerings and can be simulated. In a 24/7 environment, the engineer can model the specific traffic hour by hour, minute by minute, or even second by second to understand when the system is most effective or constrained. Moreover, the engineer can estimate the system performance of organizational partners to predict the responsiveness of interorganizational business processes. In related work, I have effectively condensed these models and used them as input to automated software-evaluation applications.³

Phase 3: Service-oriented software-integration engineers

In phases 1 and 2, service-oriented software engineers try to understand long-standing supply chains or workflows of Web services. However, real-time, on-demand composition is the future for SCSSs, so services must be well represented with quality-of-service attributes. Service-oriented software-integration engineers specify service-level agreements (SLAs) between organizational entities and manage or govern the automated composition. Their role overlaps with the software engineer's process-modeling role, but these engineers must also make real-time design and implementation decisions affecting the system's continuing operation. These engineers have two major responsibilities:

Real-time, on-demand composition is the future for SCSSs, so services must be well represented with QoS attributes.

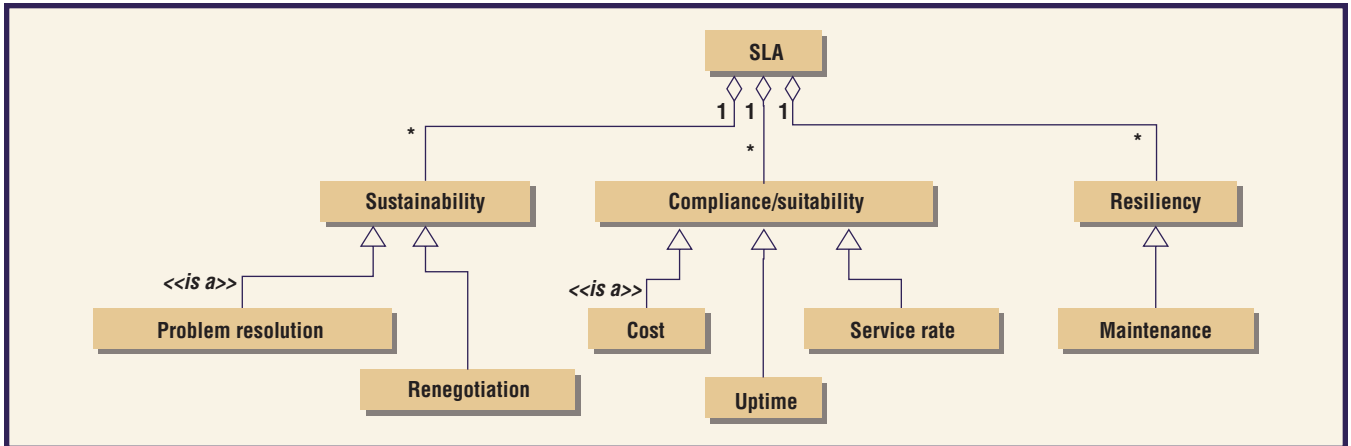


Figure 5.
Principle-based
service-level
agreement metrics.

- analyze, understand, and specify SLAs for existing services and
- develop and manage automated approaches for real-time composition.

Specifying SLAs. An SLA is a technical contract between two businesses: a producer and a consumer. It captures the agreed-upon terms between these two organizations with respect to QoS and other related concerns. In a service-oriented computing environment, a producer organization exposes implemented Web services to share capabilities. The SCSS goal is then to have consumers access these shared capabilities on demand. New standards, such as the Web Service Level Agreement and Web Service Agreement specifications⁶ enable SLAs to be associated to an individual service or to groups of Web services. These XML-based documents simplify the capture and management of SLA metrics.

The service-oriented software-integration engineer's core responsibility is to manage SLAs and incorporate them into automated systems that govern real-time performance in SCSS environments. In phases 1 and 2, service-oriented software engineers capture and estimate performance values. In phase 3, service-oriented software-integration engineers derive SLA-based guarantees of services based on those specifications and incorporate tools to govern them. I stratify the SLA criteria by introducing three principles associated with composing SLAs in SCSS environments: *compliance (suitability)*, *sustainability*, and *resiliency* (see figure 5). As a consultant, I evaluated these principles in a government cross-organizational setting.

Compliance (suitability) ensures that the consumer receives the requested composite capability at the required service level. Consider-

ing SLA terms, the composition process must ensure that the aggregate *cost*, *uptime*, and *service rate* comply with the user requirements. Cost is the sum of the prices of all services participating in the solution process. Uptime is a guarantee by the service providers that their services will be available a specified percentage of the time per day or month. Finally, service rate is the average time to complete the process, determined by totaling each service's average response times in the process.

Sustainability is the ability to maintain the underlying services in a timely fashion. Negotiation and renegotiation as well as problem resolution strongly correlate with sustainability. A consumer will require assurance that a particular business can agree on contract terms (negotiation and renegotiation) in a timely manner. In addition, the service providers must be able to resolve high-impact problems (perhaps identified by the consumer) in a timely manner. Both negotiation and problem-resolution times ensure that a consumer can meet end-user demands.

Resiliency recognizes that a service should perform at high levels over an extended period of time. If a service is frequently taken offline for maintenance, or if the frequency of updates impedes the predictability of its operation, then that service isn't very resilient. Consumers will need adequate notice before maintenance downtimes. In addition, resiliency dictates a low frequency of maintenance downtime. Uptime and maintenance time are independent.

Service composition using SLAs. Service-oriented integration engineers must manage an integrated process of service composition coupled with workflow-based SLA measures. This process has three steps: composition, evaluation, and opti-

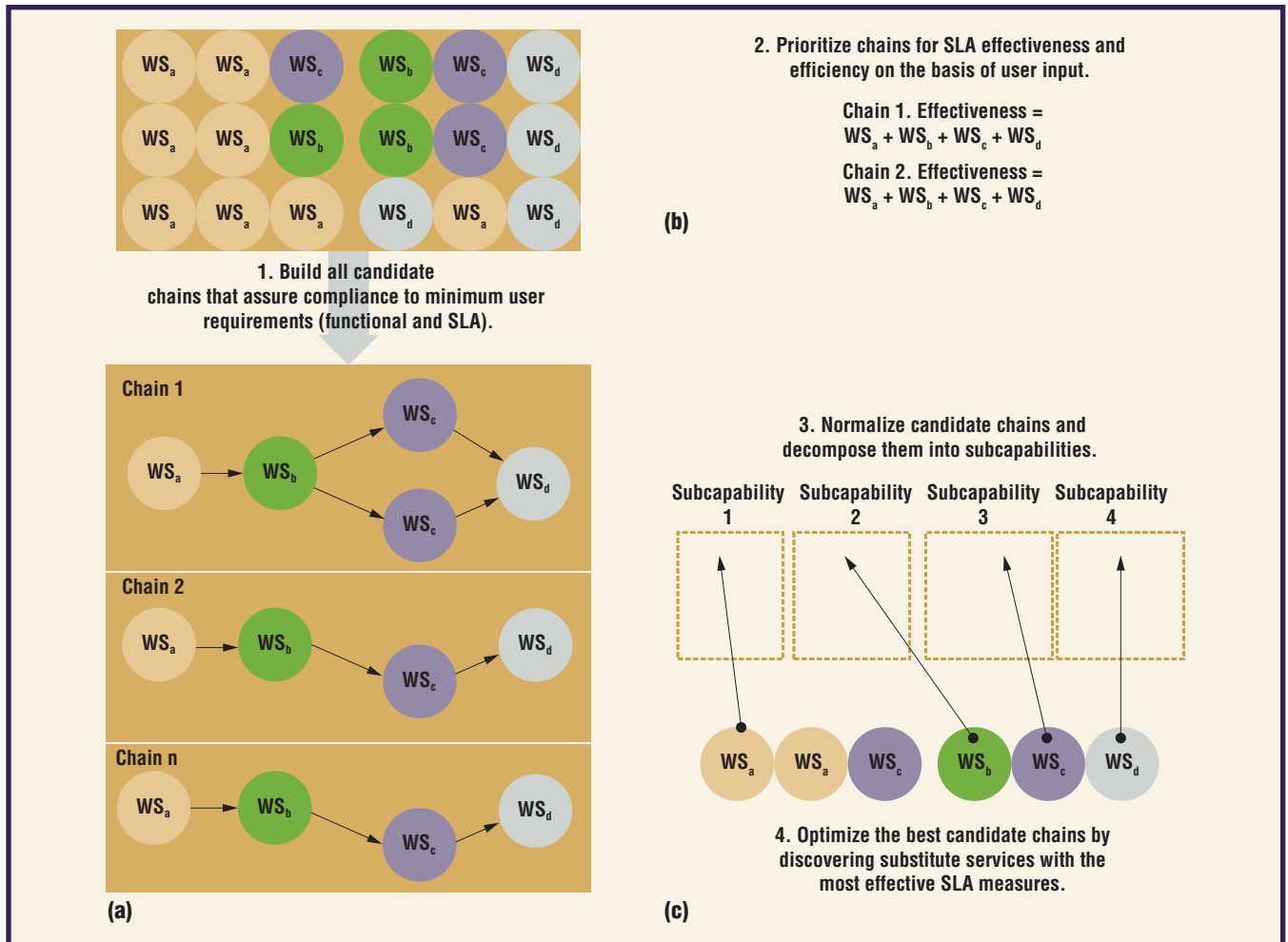



Figure 6. Integrative SLA-based composition in SCSS:
(a) composition,
(b) evaluation, and
(c) optimization.

mization. The composition step is similar to related research in Web service composition. Web services are connected via forward- or backward-chaining to match inputs to outputs until the required result is achieved (see figure 6a). A variation in this work is that the engineers also validate minimum service levels during this step. For example, if one Web service exceeds the minimum requirements for an entire chain, then that Web service must not be considered in candidate chains.

During evaluation, user priorities are employed to prioritize the list of candidate service chains (see figure 6b). Numerous dynamic-programming techniques can be used to achieve this step.⁷

However, real-time optimization in the SCSS environment is an open problem. Optimization attempts to replace services at the subprocess level to generate a “best” service chain (see figure 6c).⁸ In real-time operations, the composition and evaluation steps are re-

quired, while the optimization step is best suited for design-time decision support.

Creating new life cycles and better educating software engineers about SOA methodologies will enhance next-generation distributed enterprise systems. Industry and research organizations will need to assimilate these enhanced software life cycles and evaluate their effectiveness at the end of each phase. Considering the heterogeneity of service-centric environments, the answer might be different for different domains. However, in the future, strategic models should be developed that help architects navigate the variations across domains while preserving the modularity of SOA environments. 

References

1. P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley, 1998.

2. M. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," *Proc. 4th Int'l Conf. Web Information Systems Eng. (WISE 03)*, IEEE CS Press, 2003, p. 3.
3. M.B. Blake, "A Lightweight Software Design Process for Web Services Workflows," *Proc. 4th IEEE Int'l Conf. Web Services (ICWS 06)*, IEEE CS Press, 2006, pp. 411-418.
4. J. Mukerji and J. Miller, *MDA Guide Version 1.0.1*, tech. report omg/2003-06-01, Object Management Group, 2003; www.omg.org/docs/omg/03-06-01.pdf.
5. W.M.P. Van der Aalst et al., "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, 2003, pp. 5-51.
6. A. Andrieux et al., *Web Service Agreement Specification (WS-Agreement)*, v. 2005/09, Global Grid Forum, 2005; www.gridforum.org/Public_Comment_Docs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf.
7. M.B. Blake and D.J. Cummings, "Workflow Composition of Service Level Agreements," *Proc. IEEE Int'l Conf. Services Computing (SCC 07)*, IEEE CS Press, 2007, pp. 138-145.

About the Author



M. Brian Blake is an associate professor and the chair at Georgetown University's Department of Computer Science, where he directs the Servicecentricity Lab. He conducts applied research in the areas of service-oriented computing and enterprise integration in collaboration with various government and industry organizations in the Washington, DC area. He received his PhD in information and software engineering from George Mason University. He's a senior member of the IEEE. Contact him at mb7@georgetown.edu.

8. L. Zeng et al., "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.*, vol. 30, no. 5, 2004, pp. 311-327.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.

ADVERTISER/PRODUCT INDEX NOVEMBER/DECEMBER 2007

Advertiser/Product	Page Number	Advertising Personnel	
Addison-Wesley	112	Marion Delaney	Sandy Brown
Amacom	111	IEEE Media, Advertising Director	IEEE Computer Society,
Basic Books	110	Phone: +1 415 863 4717	Business Development Manager
Cambridge University Press	109	Email: md.ieeemedia@ieee.org	Phone: +1 714 821 8380
Classified Advertising	17		Fax: +1 714 821 4010
ESRI	7		Email: sb.ieeemedia@ieee.org
IEEE Computer Society Membership	113	Marian Anderson	
Rose-Hulman Institute of Technology	11	Advertising Coordinator	
Seapine Software, Inc.	Cover 4	Phone: +1 714 821 8380	
University of Groningen	8	Fax: +1 714 821 4010	
		Email: manderson@computer.org	
<i>Boldface denotes advertisements in this issue.</i>			
Advertising Sales Representatives			
Mid Atlantic (product/recruitment)	Southwest (product)	Midwest (product)	Southeast (product)
Dawn Becker	Steve Loerch	Dave Jones	Bill Holland
Phone: +1 732 772 0160	Phone: +1 847 498 4520	Phone: +1 708 442 5633	Phone: +1 770 435 6549
Fax: +1 732 772 0164	Fax: +1 847 498 5911	Fax: +1 708 442 7620	Fax: +1 770 435 0243
Email: db.ieeemedia@ieee.org	Email: steve@didierandbroderick.com	Email: dj.ieeemedia@ieee.org	Email: hollandwfh@yahoo.com
New England (product)	Northwest (product)	Will Hamilton	Japan
Jody Estabrook	Lori Kehoe	Phone: +1 269 381 2156	Tim Matteson
Phone: +1 978 244 0192	Phone: +1 650 458 3051	Fax: +1 269 381 2556	Phone: +1 310 836 4064
Fax: +1 978 244 0103	Fax: +1 650 458 3052	Email: wh.ieeemedia@ieee.org	Fax: +1 310 836 4067
Email: je.ieeemedia@ieee.org	Email: l.kehoe@ieee.org	Joe DiNardo	Email: tm.ieeemedia@ieee.org
New England (recruitment)	Southern CA (product)	Phone: +1 440 248 2456	Europe (product)
John Restchack	Marshall Rubin	Fax: +1 440 248 2594	Hilary Turnbull
Phone: +1 212 419 7578	Phone: +1 818 888 2407	Email: jd.ieeemedia@ieee.org	Phone: +44 1875 825700
Fax: +1 212 419 7589	Fax: +1 818 888 4907	Southeast (recruitment)	Fax: +44 1875 825701
Email: j.restchack@ieee.org	Email: mr.ieeemedia@ieee.org	Thomas M. Flynn	Email: impress@impressmedia.com
Connecticut (product)	Northwest/Southern CA (recruitment)	Phone: +1 770 645 2944	
Stan Greenfield	Tim Matteson	Fax: +1 770 993 4423	
Phone: +1 203 938 2418	Phone: +1 310 836 4064	Email: flyntom@mindspring.com	
Fax: +1 203 938 3211	Fax: +1 310 836 4067	Midwest/Southeast (recruitment)	
Email: greenco@optonline.net	Email: tm.ieeemedia@ieee.org	Darcy Giovino	
		Phone: +1 847 498-4520	
		Fax: +1 847 498-5911	
		Email: dg.ieeemedia@ieee.org	