Service-Oriented Computing: State of the Art and Research Challenges

Michael P. Papazoglou, Tilburg University
Paolo Traverso, Istituto per la Ricerca Scientifica e Tecnologica
Schahram Dustdar, Vienna University of Technology
Frank Leymann, University of Stuttgart

Service-oriented computing promotes the idea of assembling application components into a network of services that can be loosely coupled to create flexible, dynamic business processes and agile applications that span organizations and computing platforms. An SOC research road map provides a context for exploring ongoing research activities.

The service-oriented computing (SOC) paradigm uses services to support the development of rapid, low-cost, interoperable, evolvable, and massively distributed applications. Services are autonomous, platform-independent entities that can be described, published, discovered, and loosely coupled in novel ways. They perform functions that range from answering simple requests to executing sophisticated business processes requiring peer-to-peer relationships among multiple layers of service consumers and providers. Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service.

Services reflect a "service-oriented" approach to programming that is based on the idea of composing applications by discovering and invoking network-available services to accomplish some task.[1] This approach is independent of specific programming languages or operating systems. It lets organizations expose their core competencies programmatically over the Internet or various networks such as cable, the Universal Mobile Telecommunications System (UMTS), xDSL, and Bluetooth using standard XML-based languages and protocols and a self-describing interface.

Web services are currently the most promising SOC-based technology.[2] They use the Internet as the communication medium and open Internet-based standards, including the Simple Object Access Protocol (SOAP) for transmitting data, the Web Services Description Language (WSDL) for defining services, and the Business Process Execution Language for Web Services (BPEL4WS) for orchestrating services.

SOC lets developers dynamically grow application portfolios more quickly than ever before by

- creating compound solutions that use internal organizational software assets, including enterprise information and legacy systems, and
- combining these solutions with external components possibly residing in remote networks.

The visionary promise of SOC is that it will be possible to easily assemble application components into a loosely coupled network of services that can create dynamic business processes and agile applications that span organizations and computing platforms.[3] Such services will go well beyond simply exchanging information—the dominating mechanism for application integration today—to accessing, programming, and integrating application services encapsulated within old and new applications.

Key to realizing this vision is the service-oriented architecture. SOA is a logical way of designing a software system to provide services either to end-user

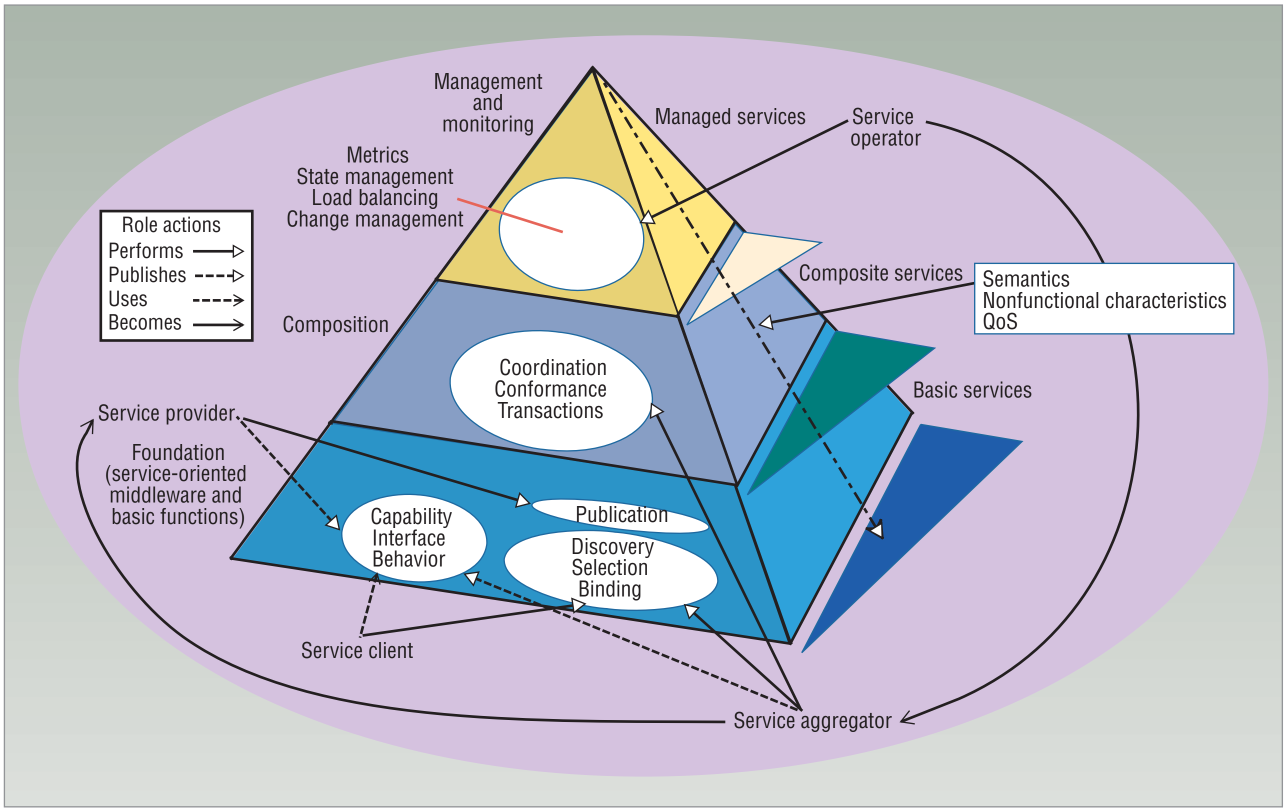

Figure 1. SOC research road map. The architectural layers provide a logical separation of functionality, while the perpendicular axis indicates service characteristics that cut across all three planes.

applications or other services distributed in a network, via published and discoverable interfaces. A well-constructed, standards-based SOA can empower an organization with a flexible infrastructure and processing environment by provisioning independent, reusable application functions as services and providing a robust foundation for leveraging these services.

SOC spans many intricately interwoven concepts, protocols, and technologies that originate in a wide range of disciplines including distributed computing systems, computer architectures and middleware, grid computing, software engineering, programming languages, database systems, security, and knowledge representation. Given this tremendous complexity, as well as the need to merge technology with an understanding of business processes and organization structures, research activities are very fragmented.

To provide the means for consolidating and streamlining current SOC research efforts, as well as prioritizing important gaps, we examine ongoing projects in the broader context of a road map that embraces four pivotal, inherently related research themes: service foundations, service composition, service management and monitoring, and service-oriented engineering. A comprehensive review of the state of the art and standards in each area identifies open problems and bottlenecks to progress.

SOC RESEARCH ROAD MAP

The SOC research road map, shown in Figure 1, introduces an extended SOA[1] that separates functionality into three planes: service foundations at the bottom, service composition in the middle, and service management and monitoring on top. This logical stratification is based on the need to separate

- basic service capabilities provided by a middleware infrastructure and conventional SOA from more advanced service functionality needed for dynamically composing services,
- business services from systems-centered services, and
- service composition from service management.

The perpendicular axis indicates service characteristics that cut across all three planes including semantics, nonfunctional service properties, and quality of service. QoS encompasses important functional and nonfunctional attributes such as performance metrics (for example, response time), security attributes, transactional integrity, reliability, scalability, and availability. Traditionally, QoS quantifies the degree to which applications, systems, networks, and other IT infrastructure elements support availability of services at a required performance level under all access and load conditions. Web services environments also demand greater availability of applications and

introduce increased complexity in terms of accessing and managing services.

The SOC research road map also defines several roles. The service requester or client and provider must both agree on the service description (WSDL definition) and semantics that will govern the interaction between them for Web services to interact properly in composite applications. A complete solution must address semantics not only at the terminology level but also at the levels that Web services are used and applied in the context of business scenarios—the business process and protocol levels. Thus, a client and provider must agree on the implied processing, context, and sequencing of messages exchanged between interacting services that are part of a business process. In addition to the classical roles of service client and provider, the road map also defines the roles of service aggregator and operator.

Finally, Figure 1 illustrates that service modeling and service-oriented engineering—service-oriented analysis, design and development techniques, and methodologies—are crucial elements for creating meaningful services and business process specifications. These are an important requirement for SOA applications that leverage Web services and apply equally well to all three service planes.

Service modeling and service-oriented engineering are crucial elements for creating meaningful services and business process specifications.

SERVICE FOUNDATIONS

The service foundations plane consists of a service-oriented middleware backbone that realizes the runtime SOA infrastructure. This infrastructure connects heterogeneous components and systems and provides multiple-channel access to services over various networks including the Internet. It lets application developers define basic service functionality in terms of the description, publishing, finding, and binding of services.

In a typical service-based scenario, a provider hosts a network-accessible software module—an implementation of a given service—and defines a service description through which a service is published and made discoverable. A client discovers a service and retrieves the service description directly from the service, possibly through metadata exchange or from a registry or repository such as UDDI. The client uses the service description to bind to the provider and invoke the service. Service provider and client roles are logical constructs, and a service can exhibit characteristics of both. Service aggregators group services provided by other providers into a distinct value-added service and can themselves act as providers.

State of the art

The requirements to provide a capable and manageable integration infrastructure for Web services and SOA are coalescing into the concept of the *enterprise service bus*.[4,5] The ESB's two key objectives are to

- loosely couple the systems taking part in the integration, and
- break up the integration logic into distinct, easily manageable pieces.

The ESB is an open-standards-based message backbone designed to enable the implementation, deployment, and management of SOA-based solutions. It is a set of infrastructure capabilities implemented by middleware technology that enable an SOA and alleviate disparity problems between applications that run on heterogeneous platforms and use diverse data formats. The ESB supports service, message, and event-based interactions with appropriate service levels and manageability. Conceptually, it has evolved from the store-and-forward mechanism found in middleware products to an enterprise application integration solution that combines, for example, application servers and integration brokers, Web services, XSLT, and orchestration technologies.[6] The ESB provides an implementation backbone for an SOA that treats applications as services.

An emerging concept for the ESB is the *container model*. In this model, a "container" for the service runtime implementation exposes the service functionalities and nonfunctional properties to the external world via the network. It establishes connectivity and message exchange patterns and provides support and facilities such as transactions, security, performance metrics, dynamic configuration, and services discovery. In addition, the container performs data and protocol adaptation, and it monitors the internal behavior and state of services.

Figure 2 shows a simplified view of an ESB that integrates a J2EE application using the Java Message Service, a .NET application using a C# client, an IBM WebSphere MQ application that interfaces with legacy applications, and external applications and data sources using Web services. A distributed query engine, normally based on XQuery or the Structured Query Language, enables the creation of data services to abstract the complexity of underlying data sources.

The ESB enables a more efficient value-added integration of numerous different application components by positioning them behind a service-oriented façade and applying Web services technology to the problem. A primary use case is for the ESM to act as the intermediary layer between a portal server and the back-end data sources that the portal server must interact with.

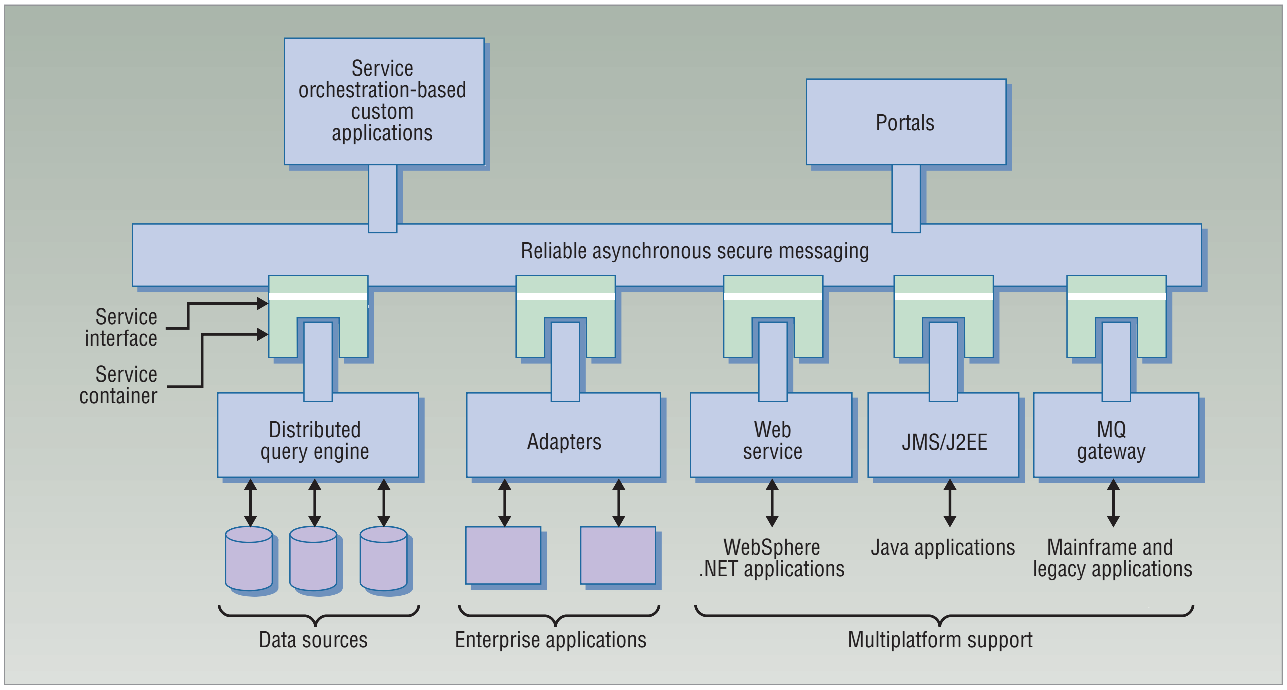

Figure 2. Enterprise service bus. The ESB connects diverse applications and technologies.

A portal is a user-facing aggregation point of various resources represented as services—for example, there can be retail, divisional, corporate employee, and business partner portals.

Research challenges

Major research challenges for service foundations in the near future include the following:

Dynamically reconfigurable runtime architectures. The services runtime infrastructure should automatically leverage distributed service components and resources to create an optimal architectural configuration according to both a particular user's requirements and the application characteristics.

End-to-end security solutions. Validating the security aspects in SOA-based applications requires a full system approach to test end-to-end security solutions at both the network and application level. The Web Services Road Map jointly developed by IBM and Microsoft has similar concerns.[7]

Infrastructure support for data and process integration. The runtime infrastructure should provide uniform consistent access to all data by all the applications that require it, irrespective of the data format, source, or location. It should also integrate service-based applications into processes and integrate processes with other processes into end-to-end constellations that span multiple institutions.

Semantically enhanced service discovery. The main challenge of service discovery is using automated means to accurately discover services with minimal user involvement. This requires explicating the semantics of both the service provider and requester. It also involves adding semantic annotations and including descriptions of QoS characteristics—for example, in the DARPA Agent Markup Language, Web Ontology Language, or other semantic markup languages—to service definitions in WSDL and then registering these descriptions. Achieving automated service discovery requires explicitly stating requesters' needs—most likely as goals that correspond to the description of desired services—in some formal request language.

SERVICE COMPOSITION

The service composition plane encompasses roles and functionality for aggregating multiple services into a single composite service. Resulting composite services can be used as basic services in further service compositions or offered as complete applications and solutions to service clients. Service aggregators accomplish this task and thus become service providers by publishing the service descriptions of the composite service they create. The aggregators also enforce policies on aggregate service invocations.

State of the art

Currently, developers widely use the terms "orchestration" and "choreography" to describe business interaction protocols that coordinate and control collaborating services.

Orchestration describes how services interact at the message level, including the business logic and execution order of interactions under control of a single end point. It is an executable business process that can result in a long-lived, transactional, multistep process model. With orchestration, one of the business parties involved in the process always controls the business-process interactions. Orchestration is achieved via BPEL4WS and other XML-based process standard definition languages.[8]

Choreography is typically associated with the public (globally visible) message exchanges, rules of interaction, and agreements that occur between multiple business-process end points rather than a specific business process executed by a single party. Service choreography is achieved via the Web Services Choreography Description Language (WS-CDL),[9] which specifies the common observable behavior of all participants.

This sharp distinction between orchestration and choreography is rather artificial, and the consensus is that they should coalesce in a single language and environment.

Service compositions must understand and respect one another's policies, performance levels, security requirements, service-level agreement (SLA) stipulations, and so on.

Research challenges

Some of the most notable research challenges for service composition in the near future include the following:

Composability analysis for replaceability, compatibility, and process conformance. Service conformance ensures a composite service's integrity by matching its operations with those of its constituent component services. It imposes semantic constraints on the component services and guarantees that constraints on data that component services exchange are satisfied. Service conformance comprises both behavioral conformance as well as semantic conformance. The former guarantees that composite operations do not lead to spurious results, while the latter ensures that they preserve their meaning when composed and can be formally validated.

Dynamic and adaptive processes. Services and processes should equip themselves with adaptive service capabilities so that they can continually morph themselves to respond to environmental demands and changes without compromising operational and financial efficiencies. In this context, the challenge is to provide techniques and support for dynamic service compositions that are self-configuring, -optimizing, -healing, and -adapting.

QoS-aware service compositions. Service compositions must be QoS-aware[10]—that is, understand and respect one another's policies, performance levels, security requirements, service-level agreement (SLA) stipulations, and so on. For example, knowing that a new business process adopts a Web services security standard from WS-Security is not enough information to enable successful composition. The client needs to know if the services in the business process actually require WS-Security, what kind of security tokens they are capable of processing, and which one they prefer.

Business-driven automated compositions. Service-oriented applications should abstract away the logic at the application or business level, such as order processing, from non-business-related aspects at the system level, such as the implementation of transactions, security, and reliability policies. This abstraction should enable the composition of distributed business processes and transactions.[11]

SERVICE MANAGEMENT AND MONITORING

When composing services, developers must be able to assess the health of systems that implement Web services as well as the status and behavior patterns of loosely coupled applications. Service management spans a range of activities, from installation and configuration to collecting metrics and tuning, to ensure responsive service execution. It typically involves gathering information about the managed-service platform, services and business processes, and managed-resource status and performance via root-cause failure analysis, SLA monitoring and reporting, service deployment, and life-cycle management and capacity planning.

Service monitoring involves monitoring events or information produced by the services and processes; monitoring instances of business processes; viewing process-instance statistics, including the number of instances in each state (running, suspended, aborted, or completed); viewing the status, or a summary, of selected process instances; and suspending, resuming, or terminating selected process instances.

State of the art

Figure 3 illustrates a conceptual Web services architecture that provides a continuous connection between the application and management channels. Manageable resources include hardware and software resources, both physical and logical—for example, software applications, hardware devices, servers, and so on. Their management capabilities are exposed as Web services that implement various management interfaces, such as those defined in the Web Services Distributed Management (WSDM) specification.

The management channel offers functionality such as availability and performance management, configuration management, capacity planning, asset protection, job control, and problem determination. A WSDL document, resource properties schema, metadata documents, and, potentially, a set of management-related policies describe a resource's management interface.

The application in Figure 3 comprises business processes that integrate basic services such as credit validation, shipping, order processing, and inventory ser-

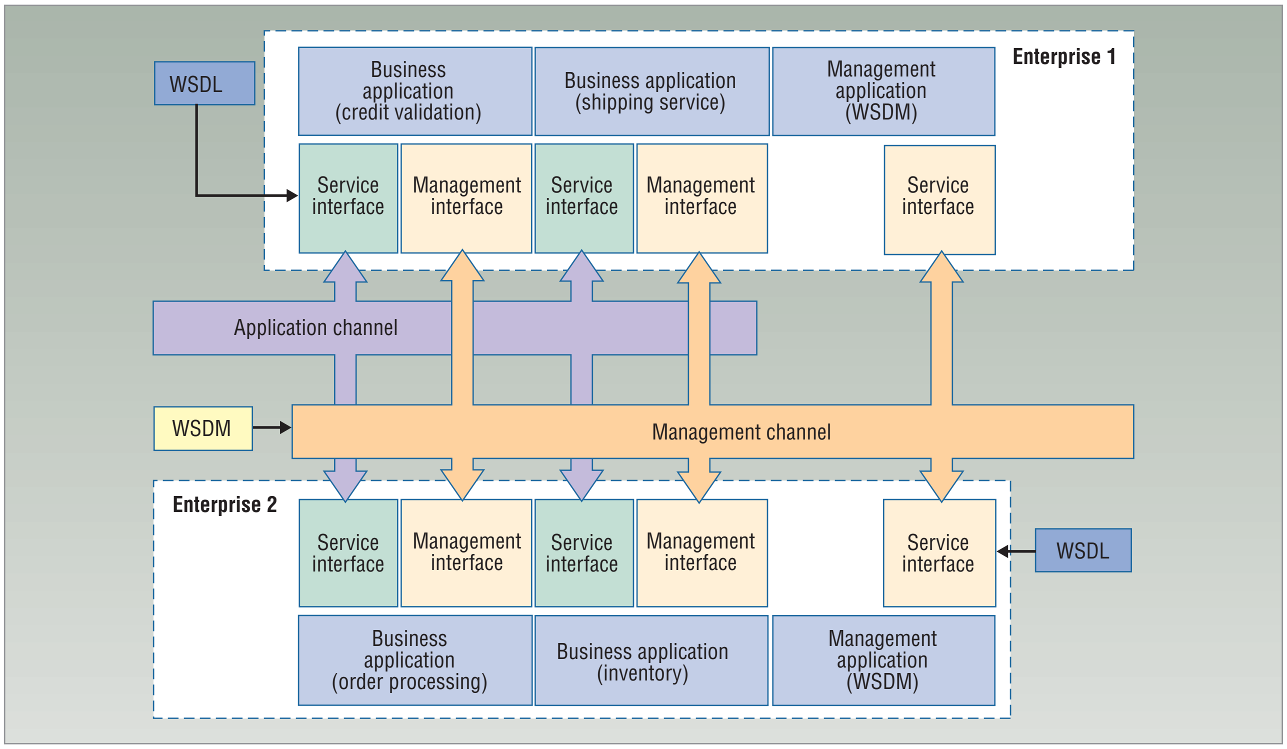

Figure 3. Web services management architecture. The architecture provides a continuous connection between the application and management channels. The application comprises business processes that integrate basic services originating from two collaborating enterprises.

vices originating from two collaborating enterprises. Resource managers provide detailed performance statistics that

- support assessment of the application's effectiveness,
- permit complete visibility into individual business processes and transactions,
- guarantee consistency of service compositions, and
- deliver status notifications when a particular activity is completed or a decision condition is reached.

To achieve consistent management of end-to-end Web services, WSDM defines a protocol for interoperability of management information and capabilities in a distributed environment via Web services. It attempts to solve distributed system management problems through two companion specifications: Management Using Web Services (MUWS) addresses the use of Web services technologies as the foundation of a modern distributed systems management framework, while Management of Web Services (MOWS) specifies the requirements for managing Web services themselves just like any other resource.[12]

Research challenges

The use of autonomic capabilities in conjunction with service-level management provides an evolutionary approach in which autonomic computing capabilities anticipate runtime system requirements and resolve problems with minimal human intervention.[3,11] Some of the most notable research challenges for the near future include the following:

Self-configuring management services. These must configure themselves automatically to adapt to different environments in which they can be installed and optimized for particular uses.

Self-adapting management services. These must adapt dynamically to changes in the environment and market using policies provided by IT professionals. Such changes could include deploying new instances of a particular kind of service or removing existing ones, or even dramatically changing runtime system characteristics.

Self-healing management services. These must discover, diagnose, and react to disruptions. They should detect system malfunctions and initiate policy-based corrective actions without disrupting the runtime environment.

Self-optimizing management services. These must monitor resources automatically and tune themselves to meet end-user or business needs. Self-optimizing management services improve overall utilization or ensure the timely completion of particular business transactions.

Self-protecting management services. These must anticipate, detect, identify, and protect against threats. Self-protecting components can detect hostile activities—such as unauthorized access and use, virus infection and proliferation, and denial-of-service attacks—as

they occur and take corrective actions to reduce their vulnerability.

SERVICE DESIGN AND DEVELOPMENT

A well-constructed SOA can empower a business with a flexible infrastructure and processing environment by provisioning independent, reusable automated business processes as services and providing a robust foundation for leveraging these services. SOAs must rely on an evolutionary software engineering approach that partly builds upon earlier processes including component-based development and business process modeling.

State of the art

Many developers think they can port existing components to Web services by creating wrappers and leaving the underlying component untouched. Because component methodologies focus on the interface, many developers assume that these methodologies apply equally well to service-oriented architectures.

The software industry now widely implements a thin SOAP/WSDL/UDDI veneer atop existing applications or components that implement the Web services, but this is insufficient for commercial-strength enterprise applications. Unless the component's nature makes it suitable for use as a Web service, and most are not, properly delivering components' functionality through a Web service takes serious redesign effort.[1]

Older software development paradigms for object-oriented and component-based development[13,14] cannot be blindly applied to SOA and Web services as they do not address SOA's key elements: services, information flows, and components realizing services.[15] These methodologies satisfy only some of the requirements of SOC applications and fail when they independently attempt to develop service-oriented solutions.

Research challenges

Major research challenges for service design and development in the near future include the following:

Engineering of service applications. SOA-based applications require a service-oriented engineering methodology[16] that enables modeling the business environment, including key performance indicators of business goals and objectives; translates the model into service design; deploys the service system; and tests and manages the deployment.[17]

Flexible gap-analysis techniques. Gap analysis purposes a business process and services a realization strategy by incrementally adding more implementation details to an abstract service/process interface. Such a strategy considers several service-realization possibilities such as green field development, top-down and bottom-up development, meet-in-the-middle development, and development based on reference models.

Service versioning and adaptivity. Developers should introduce techniques to analyze business processes in detail instantaneously, discover and select suitable external services, detect problems in service interactions, search for alternative solutions, monitor service-execution sequences step by step, and appropriately upgrade and version services.

Service governance. Due to the cross-organizational nature of end-to-end business processes composed from various service fragments that different organizations must maintain separately, service governance is a challenging issue. The potential composition of services into business processes across organizational boundaries can function properly and efficiently only if the services are effectively governed for compliance with QoS and policy requirements. Services must meet the functional and QoS objectives within the context of the business unit and the enterprises within which they operate.

Service-oriented computing is a vast and enormously complex subject, embracing many technologies that must be integrated in an intricate manner. By focusing on the inherently related themes of service foundations, service composition, service management and monitoring, and service design and development, researchers can make requisite connections among diversified activities, find links in hitherto neglected spheres of work, and align ongoing and future projects in a more meaningful and coherent manner. This in turn will lead to a harmonization of research expertise and more interoperable SOC solutions. ■

References

1. M.P. Papazoglou, *Web Services: Principles and Technology,* Prentice Hall, 2007.
2. S. Weerawarana et al., *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More,* Prentice Hall, 2005.
3. F. Leymann, "Combining Web Services and the Grid: Towards Adaptive Enterprise Applications," *Proc. CAiSE 05 Workshops,* vol. 2, FEUP Edições, 2005, pp. 9-21.
4. D. Chappell, *Enterprise Service Bus,* O'Reilly Media, 2004.
5. F. Leymann, "The (Service) Bus: Services Penetrate Everyday Life," *Proc. 3rd Int'l Conf. Service-Oriented Computing* (ICSOC 2005), LNCS 3826, Springer-Verlag, 2005.
6. M.P. Papazoglou and W-J. van den Heuvel, "Service-Oriented Architectures: Approaches, Technologies and Research Issues," *VLDB J.,* vol. 16, no. 3, 2007, pp. 389-415.
7. "Security in a Web Services World: A Proposed Architecture and Roadmap," v1.0, 7 Apr. 2002, IBM/Microsoft; http://msdn2.microsoft.com/en-us/library/ms977312.aspx.
8. T. Andrews et al., "Business Process Execution Language for Web Services," v1.1, 5 May 2003, IBM developerWorks; www.ibm.com/developerworks/library/specification/ws-bpel.

9. N. Kavantzas, "Web Services Choreography Description Language 1.0," editor's draft, 3 Apr. 2004, W3C; http://lists.w3.org/Archives/Public/www-archive/2004Apr/att-0004/cdl_v1-editors-apr03-2004-pdf.pdf.
10. F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," *Proc. IEEE Int'l Conf. Web Services* (ICWS 06), IEEE Press, 2006, pp. 205-212.
11. M.P. Papazoglou and B. Kratz, "Web Services Technology in Support of Business Transactions," *Service-Oriented Computing and Applications J.,* vol. 1, no. 1, 2007, pp. 51-63.
12. H. Kreger et al., "Management Using Web Services: A Proposed Architecture and Roadmap," v1.0, 2 June 2005, Computer Assoc./HP Development Co./IBM; ftp://www6.software.ibm.com/software/developer/library/ws-mroadmap.pdf.
13. F. Bachmann et al., *Volume II: Technical Concepts of Component-Based Software Eng.*, 2nd ed., tech. report CMU/SEI-2000-TR-008 ESC-TR-2000-007, Software Eng. Inst., Carnegie Mellon University, May 2000.
14. P. Herzum and O. Sims, *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise,* John Wiley & Sons, 2000.
15. A. Arsanjani, "Service-Oriented Modeling and Architecture," 9 Nov. 2004, IBM developerWorks; www-106.ibm.com/developerworks/library/ws-soa-design1.
16. U. Zdun and S. Dustdar, "Model-Driven Integration of Process-Driven SOA Models," to appear in *Int'l J. Business Process Integration and Management,* 2007.
17. C. Ghezzi, "Service-Oriented Computing: Where Does It Come From? A Software Engineering Perspective," keynote address, 3rd Int'l Conf. Service-Oriented Computing, Amsterdam, Dec. 2005.

Michael P. Papazoglou is the chair of computer science and director of the Infolab in the Department of Information Systems and Management at Tilburg University, Tilburg, the Netherlands. His research interests include distributed systems, service-oriented computing, enterprise application integration, and e-business technologies and applications. Papazoglou received a PhD in computer systems engineering from the University of Edinburgh. He is an IEEE Computer Society Golden Core member and distinguished visitor. Contact him at mikep@uvt.nl.

Paolo Traverso is director of information technology at the Fondazione Bruno Kessler, Istituto per la Ricerca Scientifica e Tecnologica, FBK-IRST, Trento, Italy. His research interests include automated planning, automated verification and synthesis, and service-oriented applications. Traverso received a Laurea in electronic engineering from the University of Genoa. He is a member of the IEEE Computer Society and the ACM. Contact him at traverso@itc.it.

Schahram Dustdar is a full professor of computer science, director of the Vienna Internet Technologies Advanced Research Lab, and head of the Distributed Systems Group of the Information Systems Institute at the Vienna University of Technology, Vienna, Austria. He is also honorary professor of information systems in the Department of Computer Science at the University of Groningen, the Netherlands. His research interests include service-oriented architectures and computing, mobile and ubiquitous computing, complex and adaptive systems, and context-aware computing. Dustdar received a PhD in business informatics from the University of Linz, Austria. He is a member of the IEEE Computer Society and the ACM. Contact him at dustdar@infosys.tuwien.ac.at.

Frank Leymann is a full professor of computer science and director of the Institute of Architecture of Application Systems at the University of Stuttgart, Stuttgart, Germany. His research interests include service-oriented computing and middleware, workflow- and business-process management, programming in the large, transaction processing, integration technology, and architecture patterns. Leymann received a PhD in mathematics from the University of Bochum, Bochum, Germany. He is a member of the German Computer Society. Contact him at frank.leymann@iaas.uni-stuttgart.de.