Common Misconceptions About Service-Oriented Architecture

Grace A. Lewis, Edwin Morris, Dr. Dennis B. Smith, Soumya Simanta, and Lutz Wrage
Software Engineering Institute

Service-Oriented Architecture (SOA) is having a major impact on the acquisition and development of software systems because of its potential for increased business agility, adaptability of applications, interoperability between systems, and reuse of legacy assets. However, organizations often make decisions on SOA adoption without carefully analyzing the implications of their decisions. This article outlines a set of common misconceptions about SOA and suggests ways to more effectively address critical SOA issues that potential users, developers, and acquisition officers may have.

You do not have to look far to become aware of the effect that SOA is having on software systems. Vendors are aggressively marketing hardware, software, tools, and services that support SOA implementation within organizations as diverse as the Department of Defense (DoD), banks, federal agencies, manufacturing companies, and health care providers. Even more significantly, customers are embracing SOA with the goal of reaching a previously unachievable level of interoperability among systems and agility in business practices.

SOA may currently be the best available solution for achieving interoperability and agility, as well as providing a technology upgrade path that preserves the investment in legacy systems and simplifies deployment of new systems. However, our experience from working with customers considering the adoption of SOA suggests that they often have a variety of misconceptions that lead them to greatly underestimate the effort required to successfully implement SOA. These misconceptions are dangerous because they make organizations more susceptible to vendor advertising and hype. In addition, these misconceptions are often embraced by internal IT organizations, leading them to over-promise new capabilities, while underestimating the cost and effort required for achieving even modest improvements. Although some of these common misconceptions also apply to traditional single systems, we focus on their relevance to SOA-based systems.

We hope that by recognizing these misconceptions, organizations can better understand and evaluate the promises of vendors and improve their own internal SOA expectations and planning processes.

Basic SOA Concepts

SOA is a way of designing systems composed of services that are invoked in a standard way. As an architectural style, SOA is neither a system architecture nor a complete system. An SOA-based system is composed of the following:

- Services that are reusable components that represent business or mission tasks, such as customer lookup, weather, sensor placement, account lookup, or credit card validation.
- Service consumers that are clients for the functionality provided by the services, such as end-user applications, systems, or even other services.
- SOA infrastructure that connects service consumers to services.

The most common approach to SOA implementation is that of Web services, which relies on common standards that include HTTP, SOAP, WSDL, and UDDI. However, other SOA-based systems can be implemented using such technologies as MOM, IBM WebSphere MQ, publish-subscribe systems such as JMS, and CORBA.

Some SOA Misconceptions

Seven common misconceptions are identified in the following subsections. The subsection heading represents a statement in the form that an organization might express it. The body of each subsection discusses why the statement expressed in the heading can be misleading. It also provides advice on how to avoid falling into common traps.

SOA Provides the Complete Architecture for a System

Chief among SOA misconceptions is the belief that simply by adopting an SOA strategy for the enterprise, an organization has established a complete well-crafted architecture that will help the organization achieve its IT goals. In reality, SOA is not an architecture, but an architectural pattern from which a number of specific architectures can be derived – both good and bad. An architectural pattern provides guidance to an architect that enables leveraging best practices for that specific pattern. It defines a set of element types, a topological layout of the elements that shows their relationships, semantic constraints on elements, and interaction mechanisms [1]. For example, the elements in the SOA pattern include service consumers, service descriptions, service implementations, and possibly a service bus. One relationship is that between service providers and service consumers. In the case of Web Services, consumers and services are connected by HTTP or HTTPS connectors carrying SOAP messages. Given the architectural elements, or building blocks, any number of systems can be developed based on this architectural pattern. These concrete elements and their interactions are the architecture of the system.

The misconception that SOA provides a complete architecture also leads customers to believe that they can buy SOA *off the shelf*. Although there are a number of products available in the marketplace that can help an enterprise implement

Acronyms

BPEL	Business Process Execution Language
CORBA	Common Object Request Broker Architecture
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IT	Information Technology
JMS	Java Messaging Service
MOM	Message-Oriented Middleware
MQ	Message Queue
OWL-S	Object Window Library for Services
QoS	Quality of Service
SAML	Security Assertion Markup Language
SLA	Service-Level Agreement
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language
WSCL	Web Services Conversation Language
WSDL	Web Service Description Language
WS-I	Web Services Interoperability
UDDI	Universal Description, Discovery and Integration

SOA, none of them are actually an implementation of an SOA-based system. Software architects still need to architect systems based on the SOA architectural pattern. They have to design services and service interactions that meet the qualities that stakeholders expect of the system. In addition, the architect(s) must make decisions on how services are implemented. Service implementations may involve developing new software, wrapping a legacy software system, incorporating services provided by third parties, or a combination of these options.

Information about the quality attributes of SOA-based software systems is just beginning to become available in the literature: One report finds that SOA promotes modifiability, interoperability, and extensibility, but can have a negative impact on security, performance, testability, and auditability [2]. For a given system, the architect needs to understand the quality attribute requirements and needs to architect a concrete system around the tradeoffs that are most important to the stakeholders of the system.

All Legacy Systems Can Be Easily Integrated Into an SOA Environment

One of the most attractive promises of moving towards SOA is that it enables reusing legacy systems, thereby providing a significant return on the investment in these systems. However, migrating legacy systems is neither automatic nor easy. It might not make business or technical sense to migrate the legacy system to an SOA environment.

It is important to understand technical constraints of the legacy components, such as immature technology, that may require significant rework. In addition, it is necessary to understand business issues, such as the business case that will justify the migration of legacy components to services in the specific context. An upfront and hands-on analysis of technical feasibility and the resultant return on investment will help to avoid last-minute surprises.

This analysis must answer at least the following questions:

1. Have consumers for the services been identified?
2. Is it technically feasible to create a service from the legacy system or part of the system?
3. How much would it cost to expose services from the legacy system?
4. What changes will have to be made to legacy systems in order to use these services?
5. How much will these changes affect the current end users of the legacy system and other dependent production systems?
6. Are the costs of exposing services, together with the associated risks of making the required changes, feasible from a business perspective?

The bottom line is that there are issues to take into consideration that go beyond adding a service interface to an existing system.

SOA Is All About Standards and Standards Are All That Is Needed

This statement primarily applies in the context of Web services, the main standards-based technology available today to realize SOA. This leads to a corollary misconception that SOA and Web services are the same. In reality, Web services are only one potential approach to SOA implementation.

It is true that public standards like those supporting Web services are often preferable to proprietary solutions because they are (potentially) supported by a wider community. But, most Web service standards are still emerging and subject to multiple interpretations.

Basic infrastructure standards that support the exchange of messages between service consumer and provider-such as HTTP, XML, XML Schema, SOAP, WSDL, and UDDI are the most developed and mature of the Web service standards. However, being stable for years does not mean that the standards are complete. For example, after adopting basic infrastructure Web service standards, some organizations found that their services still could not communicate information effectively with other services due to different design decisions and flexibility in the standards. The WS-I Basic Profile was constructed to provide better interoperability across implementations using basic infrastructure standards [3]. In addition, revisions to standards are likely in any area undergoing rapid advances in technology.

Standards for service composition (e.g. WSCL, WS-Coordination, BPEL, and cross-cutting standards [e.g. WS-Security, SAML, WS-Transaction, WS-Reliability]) are less mature and far less stable than basic infrastructure standards. Currently, there are a number of competing proposals and standards for service composition and cross-cutting concerns that conflict and overlap. Regarding these less mature areas of Web services, the old saying sums it up – *the best thing about standards is that there are so many to choose from.*

SOA Is All About Technology

Vendors pushing SOA products will (for good reason) promote their technologies as the solution to an organization's IT problems. However, SOA also entails changes to the organization's IT governance model – the set of rules and regulations under which an IT department operates, and the mechanisms to ensure compliance with those rules and regulations. This is especially true if SOA is used to support business processes or mission threads. Therefore, a well-defined governance model that includes items such as the following is essential for the success of SOA implementation:

- Service identification that maps to business or mission goals.
- Service repository management.
- Service implementation guidelines.
- Change management to deployed services.
- Mechanisms, tools, and policies for maintaining and monitoring deployed services.
- Policy enforcement at design and run time.
- Security and access control.
- Definition and enforcement of SLAs between service consumers and providers.

The implementation of SOA in an organization should be part of a larger effort to assure that SOA and related governance are aligned with strategic goals and objectives.

The Use of Standards Guarantees Interoperability in an SOA Environment

True interoperability can only be achieved if service consumers and providers interoperate at both the syntactic and semantic levels. There is interoperability at the syntactic level if they can exchange raw data elements such as text, numbers, or dates. There is interoperability at the semantic level if they understand and agree on the meaning of exchanged data. For example, a spacecraft monitoring application may rely on a service that does an analysis of data received from onboard sensors. The service may correctly perform the analysis of the raw temperature data. However, it may make an assumption that the temperature data is expressed in Celsius as opposed to Fahrenheit. In such a case, there is interoperability at the syntactic level, but not at the semantic level. In this example, both the requesting consumer and the onboard sensor share a common understanding that the number exchanged represents temperature. However, there

must also be a deeper understanding of the meaning of that value, such as the temperature unit or where and how it was measured [4]. The results of an incorrect assumption in this case could prove disastrous for the mission.

In the case of Web services, for service consumers and providers to be interoperable it is not sufficient to agree on the representation of data in XML documents because there is no way to specify the meaning of data in an XML or WSDL document other than in text descriptions. The problem is that text descriptions are imprecise, are often not filled in, and are not readable by machines, rendering them open to multiple interpretations by human developers. Also, even though the full XML Schema Datatypes specification can be used to specify data, it is rare to see anything other than a data type in the WSDL document that describes Web service operations. Optimal methods of describing the meaning of Web service inputs and outputs in a formal manner is still an area of active research [5, 6].

A Service Registry Allows Service Binding Dynamically at Runtime

Currently, binding to services is usually done at design time. This is referred to as static binding or fully-grounded binding. Discovery and composition of services are done at design time such that the developer can discover the syntax and semantics of the service before it is actually used. In the case of dynamic binding, discovery and composition of services are done at runtime. This is currently a complex and poorly supported task.

In a basic scenario of dynamic binding, service consumers retrieve the service address from a registry before each call to the service. If there are several providers of the same service, the service consumer can choose at runtime which one to use. The consumer can also rank providers based on quality of service criteria, choose a preferred provider, and use others as backup if the preferred service is not available.

More advanced automatic discovery and composition of new services at runtime requires the use of common ontologies by service providers and consumers within a domain to describe function and usage of services. Given this shared ontology, it would still be necessary to develop components that can construct the right queries for the discovery of services, compose services when there is not a single service that provides the needed function, and then provide the right data to invoke the discovered service. Current technologies have not advanced to a point where this is possible in production environments [7].

Testing SOA-Based Systems Is No Different Than Testing Any Other Type of System

Testing service consumers, as well as the services themselves, is challenging for various reasons. Most traditional testing techniques cannot be directly applied to services in the SOA world because testing has to occur at runtime and in real time [8]. Independent testing of a service from a service provider perspective is different from that of a service consumer. Moreover, the service provider and consumer must collaborate and cooperate to ensure correctness and trustworthiness of services [8].

"Modeling and simulation can provide some guidance and confidence during the design phase, but they are not a substitution for end-to-end testing of service-based applications."

Service consumers can only be fully tested when the invoked services (or test instances of them) are available. The ease of testing will most likely depend on whether the service is internal or external to the organization – there is more control if it is internal.

In an SOA environment it is common for different services to be owned by different organizations and for services to use different technologies. Because an SOA environment is distributed, loosely coupled, and asynchronous, testing can be significantly more complex than simply testing a set of known paths in a single system [9]. Modeling and simulation can provide some guidance and confidence during the design phase, but they are not a substitution for end-to-end testing of service-based applications [9]. Service consumers will necessarily have to be prepared to deal (or not to deal) with degraded service modes and complete service failure.

Services can be reused across applications that cross enterprise boundaries. Changes requested by one service consumer in an existing service can result in undesired results for another service consumer. Changes in service interface and implementation must be tested continuously by each of the service consumers in order to ensure that the actual service behavior conforms to intended behavior. Finally, service providers have to extensively test their services because they cannot anticipate all the possible scenarios in which their service will be used. Testing has to cover functionality, load testing and stress testing, as well as other elements specified in an SLA.

Conclusions

We believe SOA may be the best current approach for achieving critical interoperability, agility, and reusability goals that are common to many organizations. However, we also believe that the difficult reality of building and managing large-scale SOA-based systems often gets lost in the understandable corporate desire for sweeping improvements and the hype of vendors.

Our intent is not to discourage organizations from adopting SOA, but to caution them about some important issues and risks to consider while creating their SOA strategy. Most of these issues are currently active areas of research in the service-oriented computing community. The solutions will require time to mature.◆

References

1. Bass, L., P. Clements, and R. Kazman. Software Architecture in Practice. Addison Wesley, 2003.
2. O'Brien, L., L. Bass, and P. Merson. Quality Attributes and Service-Oriented Architectures. Pittsburgh: Software Engineering Institute (SEI), Carnegie Mellon University (CMU), 2005.
3. Web Services Interoperability Organization. "Basic Profile Version 1.1." 2004. <www.ws-i.org/Profiles/BasicProfile-1.1.html>.
4. Lewis, G., and L. Wrage. "Case Study in COTS Product Integration Using XML." Proc. of the Third International Conference on COTS-Based Software Systems, Redondo Beach, CA, Feb. 1-4, 2004: 41-52.
5. Martin, D. et al. "Bringing Semantics to Web Services: The OWL-S Approach." Proc. of the First International Workshop on Semantic Web Services and Web Process Composi-

tion. July 6-9, 2004, San Diego, CA.
6. Roman, D., et al. "Web Service Modeling Ontology." Applied Ontology 1.1 (2005).
7. Metcalf, C., and G. Lewis. Model Problems in Technologies for Interoperability: OWL Web Ontology Language for Services (OWL-S). Pittsburgh: SEI, CMU, 2006.
8. Tsai, W. T., et al. "Cooperative and Group Testing in Verification of Dynamic Composite Web Services." Computer Software and Applications Conference, Sept. 2004.
9. Acharya, M., et al. "SOA in the Real World-Experiences." Lecture Notes in Computer Science, Volume 3826, Nov. 2005, pp. 437-449.

About the Authors

Grace Lewis is a senior member of the technical staff at SEI where she currently leads the system of systems engineering team within the Integration of Software-Intensive Systems (ISIS) initiative. Her current interests and projects are in SOA, legacy system modernization, and software development life-cycle activities in systems of systems. She has a bachelor's degree in systems engineering and an executive master's of business administration from Icesi University in Cali, Colombia, and a master's degree in software engineering from CMU.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-5851
Fax: (412) 268-5758
E-mail: glewis@sei.cmu.edu

Soumya Simanta is a member of the technical staff at the SEI, where he is part the ISIS initiative. His current research is focused on system of systems engineering, service-oriented architecture, modernization of legacy systems, grid architecture, and evaluation of current technologies that support integration and interoperability between systems. Previously, he did software design and development in the finance and telecom domains. Simanta has a bachelor's degree in electronics engineering from Sambalpur University, and a master's degree in software engineering from CMU.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-7602
Fax: (412) 268-5758
E-mail: ssimanta@sei.cmu.edu

Edwin Morris is a senior member of the technical staff at SEI. He has more than 20 years experience in software, including design and development of embedded real time operating systems and tools, management of technical staff, and support for military, government, and corporate software initiatives. Morris is currently a member of the ISIS initiative.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-5754
Fax: (412) 268-5758
E-mail: ejm@sei.cmu.edu

Lutz Wrage is a senior member of the technical staff at SEI where he is part of the Dynamic Systems program. His research includes work on SOA and systems-of-systems engineering. Before joining the SEI, he worked in the area of Enterprise Resource Planning systems integration and customization. Lutz holds a degree in computer science from the Technical University-Berlin and a master's degree in software engineering from CMU.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-7771
Fax: (412) 268-5758
E-mail: lwrage@sei.cmu.edu

Dennis Smith, Ph.D., is a senior member of the technical staff and lead of the ISIS initiative at the SEI. This initiative focuses on developing and applying methods, tools, and technologies that enhance the effectiveness of complex networked systems and systems of systems. He has been involved with working with DoD organizations in developing an SOA capability, including issues of SOA strategy, governance and migration of legacy assets to SOA. Previously, he was a member of the Product Line Systems Program and technical lead in the effort for migrating legacy systems to product lines. He has published a variety of books, articles and technical reports, and has given talks and keynotes at conferences and workshops. He was the co-editor of the Institute of Electronics and Electrical Engineers and International Organization for Standardization-recommended practice on computer-aided software engineering adoption, and has been general chair of two international conferences. Smith holds a masters and a doctorate degree from Princeton University, and a bachelor's degree from Columbia University.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-6850
Fax: (412) 268-5758
E-mail: dbs@sei.cmu.edu