Is SOA Being Pushed Beyond Its Limits?

Grace A. Lewis
Software Engineering Institute, Carnegie Mellon University, United States

Abstract
This article presents some of the characteristics of future service-oriented systems. Also, it focuses on a set of architecture and design drivers for these future service- oriented systems that can help meet new expectations without sacrificing the loosely coupled, stateless, standards-based characteristics that have driven SOA adoption in many contexts. The article concludes with thoughts on the key role of the architect in the service-oriented systems-development process.

Keywords: *service-oriented systems*

1. Introduction

It is clear that service-oriented architecture (SOA) is having a substantial impact on the way in which software systems are developed. According to a 2007 Gartner Group report, 50 percent of new mission-critical operational applications and business processes were designed in 2007 around SOA, and that number will be more than 80 percent by 2010. Despite recent news that SOA adoption rates are falling and that “SOA is dead,” Forrester Group recently reported that SOA adoption is increasing across all of its vertical-industry groups. The reality is that SOA is currently the best option available for systems integration and leverage of legacy systems.

SOA is a way of designing, developing, deploying, and managing systems, and it is characterized by coarse-grained services that represent reusable business functionality. Service consumers compose applications or systems by using the functionality that services provide through standard interfaces.

At a high level:
- Services provide reusable business functionality.
- Service consumers are built by using the functionality from available services. Service-interface definitions are first-class artifacts.
- There is a clear separation between the service interface and service implementation that come from the legacy systems, external systems, or code that was built specifically for this purpose.
-An SOA infrastructure enables the discovery, composition, and invocation of services.
-Protocols are predominantly, but not exclusively, message-based document exchanges.

From a more technical point of view, SOA is an architectural style or design paradigm; it is neither a system architecture nor a complete system. As an architectural style, it is characterized by a set of components and connectors, situations in which the style is applicable, and benefits that are associated with implementing the style.

If it is implemented correctly, SOA adoption can provide business agility, reuse of business functionality, and leverage of legacy systems for an organization. Many organizations recognize these potential benefits and are adopting SOA—some more successfully than others. SOA has indeed “crossed the chasm,” [1] according to a recent Software AG user survey in which 90 percent of the respondents claim to have made some commitment to SOA adoption [2].

However, as with any technology, as SOA is adopted within organizations and becomes a mainstream paradigm for systems development, the requirements and expectations that are placed on service orientation increase. What was initially an approach for asynchronous document-based message exchanges now has performance, availability, reliability, security and other expectations of traditional distributed systems. As a result, the loosely coupled, stateless, standards-based nature of the relationship between service consumers and service providers in service-oriented systems is changing, so as to meet these new requirements. In addition, global enterprises and the emerging market of third-party services that are being made available through the Cloud are also placing expectations on service-oriented system architecture and design.

The first part of this article presents some of the characteristics of future service-oriented systems. The second part focuses on a set of architecture and design drivers for these future service-oriented systems that can help meet new expectations without sacrificing the loosely coupled, stateless, standards-based characteristics that have driven SOA adoption in many contexts. Finally, the article concludes with thoughts on the key role of the architect in the service-oriented systems-development process.

2. Future Service-Oriented Systems

Between 2005 and 2007, multiple surveys were conducted by organizations such as Forrester, Gartner, and IDC that showed that the top drivers for SOA adoption were mainly internally focused: application integration, data integration, and internal process improvement. This fact is changing. A recent survey published by Forrester shows that the number of organizations that are currently using SOA for external integration is approximately one third of the surveyed organizations [3]. While the percentage of externally focused SOA applications is still a minority, this percentage has been growing, and the trend will continue as organizations look at SOA adoption for supply-chain integration, access to real-time data, and cost reduction through the use of third-party services via the Cloud or Software as a Service (SaaS). As organizations expand their systems to cross organizational boundaries, the requirements on their systems also expand—from consumer, provider, and infrastructure perspectives. What follows are some requirements that will be typical of these future (or even current) service-oriented systems.

2.1 Security

The security threats for service-oriented systems are not new or different; it is the level of exposure that is greater. Service-oriented systems have an unknown and dynamic attack surface. Attack surface refers to the set of ways in which an adversary can exploit vulnerabilities and potentially cause damage. An attack surface can be measured in terms of three kinds of resources that are used in attacks on the system: methods (for example, an API), channels (for example, sockets), and data (for example, input parameters). The greater the number of resources that are accessible for attack, the greater the attack surface and, therefore, the more insecure the software environment [4]. From a more global perspective of security, issues such as identity management, dynamic secure-service composition, and trust in third-party services become important requirements in this type of system.

2.2 Runtime Monitoring and Adaptation

Runtime monitoring of systems is a common practice for determining the health of a system. SOA infrastructures can be configured to gather certain measures during system execution, and tools can be integrated into the system to produce reports and alerts if measures cross certain thresholds. Runtime adaptation refers to the capability of the system to adjust itself at runtime when these thresholds are crossed, so as to continue to meet quality requirements. For example, a system might start an additional instance of a service under particular load conditions or restrict access to a service if there is a suspicion that the security of the system has been compromised. These actions are possible when there is full control over a system; however, when services belong to third parties, there is much less control, and it becomes difficult to weave and consolidate the different logs that are emitted from different sources to paint an overall picture of the system.

2.3 Dynamic Binding

The word dynamic is often used to describe the binding between service consumers and services. There are various degrees of dynamism. At the lower end of the spectrum is late binding of a proxy service to a specific service instance that depends on user context or load-balancing policies. At the higher end of the spectrum is fully dynamic binding in which service consumers are capable of querying service registries at runtime, selecting the "best" service from the list of returned services, and invoking the selected service—all at runtime, and without human intervention. Late binding is a common, out-of-the-box feature of many commercial and open-source SOA infrastructures, such as an enterprise service bus (ESB). Fully dynamic binding, on the other hand, requires semantically described services that use an ontology that is shared between service consumers and service providers. Semantic Web Services represent an active area of research, as well as an unsolved problem that is not yet ready for large-scale deployment.

2.4 Multiple Consumers and Consumer Devices

As service-oriented systems start crossing organizational boundaries, the variety of service consumers will increase. Services will have to deal with heterogeneous service-consumer development and computing platforms. The proliferation and increasing power of handheld devices, along with the need for access to real-time data, are driving business applications to run on resource-constrained devices such as handheld devices, PDAs, and cell phones. In the case of third-party service providers, the fact that service consumers might be unknown adds an additional requirement of anticipating potential consumer profiles and usage patterns.

2.5 Coexistence with Other Architectural Paradigms and Technologies

Because Web Services are the main standards-based technology that is available today for implementation of service-oriented systems, a common misconception is that Web Services and SOA are the same. In fact, Web Services are only one potential approach to SOA implementation. In a traditional Web Services environment, service consumers interact with services via XML-messages that are encoded by using SOAP over HTTP in a request/response manner. While this is appropriate in many contexts, especially in enterprise contexts, it might not be appropriate in other contexts of high-performance or real-time requirements. For example, certain business processes or real-time workflows might be too dynamic and complex to be modeled by traditional sequential processing methods. In this case, event-driven SOA provides a potential solution by combining the traditional SOA request/response paradigm with the event-driven architecture (EDA) event publish/subscribe paradigm.

Another example is high-performance and real-time systems that are usually tied to requirements for higher information bandwidth, as well as much lower latencies or delays on the information. As demands become more real-time, the need for performance, predictability, and load balancing tips the scale towards point-to-point (P2P), tightly coupled architectures, as opposed to more loosely coupled architectures. Common SOA implementations that are based on HTTP, such as Web Services, might not be acceptable, because HTTP is not reliable, has limited bandwidth, introduces very high latencies, and cannot buffer, queue, and deliver messages to systems that are either temporarily unavailable or will join at a later time. For this reason, real-time support in SOA environments focuses on EDAs and publish/ subscribe systems as a way to support real-time requirements, yet maintain the loosely coupled nature of service-oriented systems.[5], [6]

As service-oriented systems depart from what is currently standardized—mainly, Web Services (whether WS* or REST)—there will be trade-offs. For example, maintainability of the system becomes more difficult when there are multiple architecture paradigms and when tool availability decreases.

2.6 Governance

The requirement for governance will not come as an explicit requirement; however, as systems start to cross organizational boundaries, the need for governance becomes even more important. SOA governance is the set of policies, rules, and enforcement mechanisms for developing, using, and evolving service-oriented systems, as well as for analysis of their business value. It includes policies and procedures, roles and responsibilities, design-time governance, and run-time governance [7], [8], [9]. Design-time governance includes elements such as rules for strategic identification of services, development, and deployment of services; reuse; and migration of legacy systems. It also enforces consistency in the use of standards, SOA infrastructure, and processes. Run-time governance develops and enforces rules to ensure that services are executed only in ways that are legal, and that important run-time data is logged. From a life- cycle point of view, design-time governance applies to early activities such as planning, architecture, design, and development. Run-time governance applies to the deployment and management of service-oriented systems. In a multi organizational environment, governance has to be extended to include policies and procedures for the identification and binding to external services and the establishment and monitoring of service-level agreements (SLAs) between service providers and consumers.

3. Architecture and Design Drivers for Future Service-Oriented Systems

The requirements for future service-oriented systems present a challenge to system architects. Ideally, the goal is to meet new expectations without sacrificing the loosely coupled, stateless, standards-based characteristics that have driven SOA adoption in many contexts. What follows are some architecture and design drivers that will have to be embedded into these systems.

3.1 Context Awareness

In a context-aware SOA environment, services can be selected and adapted every time in accordance with the user and invocation- context requirements and profiles—for example, provision of a service that:

-Has different performance, reliability, or security characteristics, according to who invokes the service and from where it is invoked.
-Returns information that is based on the language, time zone, and invocation environment of the user.
-Returns different views of data, depending on the characteristics of the device from where it is invoked.

To enable loose coupling between service consumers and services, the system architecture will have to abstract the complexity and multiplicity of implementation options. Architects will have to make trade-offs, such as whether services will expose a single standardized interface and a robust infrastructure will handle all of the necessary transformations and routing, or whether multiple service interfaces are exposed, which places fewer requirements on the infrastructure (probably, at the expense of maintainability). From a technology perspective, there is currently no standard for representing user context, which means that design decisions must be made to determine when and how user-context information is obtained [8].

3.2 Instrumentation for Runtime Monitoring and Adaptation

If a service-oriented system includes runtime monitoring and adaptation, all system elements must be instrumented so as to gather the right measures and receive the proper "orders" on what to do when thresholds are crossed. From an architecture and design perspective, this translates into architectural constructs for measurement and instrumentation in the SOA infrastructure, services, and even service consumers. Ideally, these constructs should be highly configurable so as to accommodate SLA changes and changes in service providers. For example, recent research shows that to design self-adaptive systems, the feedback loops that control self-adaptation must become first-class entities at the expense of added complexity [9].

3.3 Service Usability

In a growing market of third-party service brokers and providers, the aspects that can make a service more or less attractive include functionality, attached SLAs, and usability. Characteristics that make a service more usable or less usable can include interface design, options in messaging protocols, add-ons (such as test cases and test instances), and any other metadata that can tell consumers more about the service. Therefore, the task of service-interface design extends beyond simply defining the messages that are exchanged between providers and consumers. For example, architectural constructs would have to be put in place to support advanced service registries, multiple messaging options, test instances, SLA monitoring, and any other characteristic that contributes to the perception of service usability.

3.4 Federation

As service-oriented systems grow in size, the centralization of certain aspects might become a bottleneck. Federation can be a solution to this problem. In this context, federation refers to predefined agreements on aspects of the system that allow the autonomy of individual components.
Some aspects of service-oriented systems that might require federation in large-scale settings are:

-Identity management. This is the aspect that is most commonly associated with federation in SOA environments. Federated identity management means that there is a cooperative contract that has been set up among multiple identity providers and uses a decentralized approach, so that an identity in one of the identity providers is recognized by other identity providers in the federation [12]. From a consumer perspective, this means not having to log in to every single system that is involved

in the execution of a particular business process or workflow. Some of the challenges of federated identity management include trust, translation among multiple standards, and synchronization.
-SOA infrastructures. In large-scale service-oriented systems that span multiple organizations, it is unlikely that all organizations will have the same SOA infrastructure. In this case, federation would allow participating organizations to maintain their SOA infrastructures, while shared aspects such as policy management and governance mechanisms are agreed upon, propagated throughout the system, and implemented locally.
-Service registries. Federated service registries allow registries to appear as a single, virtual registry and individual organizations to retain local control over their own registries.
Regardless of the aspect of the system that is federated, there will need to be architectural constructs for establishing agreements, virtualization, and synchronization upon changes.

3.5 Automated Governance

The key to governance implementation is adding control to a system without creating a lot of extra work to its developers and users.
The approach is governance automation. The burden of ensuring compliance and enforcement gets pushed to the SOA infrastructure. There are tools and SOA infrastructures in which some governance automation is built in; in the end, however, the goal is the ability of an organization to ensure that development and deployment adhere to its own policies and standards, which might not be what is codified in existing tools.
Some aspects of governance that can be automated are:

-Workflows for service identification.
-Service-deployment procedures.
-Compliance with regulations such as the Health Insurance Portability and Accountability Act (HIPAA) and Sarbanes-Oxley.
-Compliance with internal security policies.
-Runtime measurements and logging.
-SLA management.

Architectural constructs will need to be developed for aspects of SOA governance that are critical for SOA implementation and are not covered (or are implemented differently) by the existing SOA infrastructure and SOA governance tools. In multiorganizational settings, the challenge is how to deal with conflicting policies and procedures among organizations.

3.6 Specialized SOA Design Patterns

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in software design. Gamma et al. produced a set of design patterns for object-oriented systems that triggered the usage of the term in software design [10].
Since then, design patterns have been produced for different types of systems, including service-oriented systems [11].
Given the expectations that are being placed on service-oriented systems, architects will have to build and research patterns to address the expectations of future service-oriented systems. This includes patterns for:

- Service orientation in multi organizational environments.
- Embedding system qualities into SOA infrastructures.
- Service-interface design.
- Integration with other technologies.

4. Conclusions

SOA is potentially being stretched beyond its limits. What was initially an approach for asynchronous document-based message exchanges now has performance, availability, reliability, security, and other expectations of traditional distributed systems. To solve this problem, multiple specifications and standards have been proposed and created, middleware products are becoming more robust, and the community has started to embrace terms such as event-driven SOA and real-time SOA. Therefore, the loosely coupled, stateless, standards-based nature of the relationship between service consumers and service providers in service-oriented systems is changing, so as to meet these new requirements. In addition, global enterprises and the emerging market of third-party services that are being made available through the Cloud are placing new expectations on service-oriented system architecture and design.
SOA is not a “one-size-fits-all” solution. As an architectural style, SOA is an appropriate solution in some situations; however, there are situations in which it is not

appropriate or it has to be used in conjunction with other technologies to meet service qualities. The architect of future service-oriented systems is going to play a crucial role in determining what expectations can or cannot be met by SOA adoption, and where trade-offs can be made for the benefit of the organization and the accomplishment of system qualities.

- Early, contextual technology evaluation — As the use of SOA for external integration and the expectations of SOA adoption increase, many promises will be made on the benefits of SOA in these scenarios that will probably not be validated until implementation. The role of the architect is to perform early, contextual technology evaluation and continuous technology scouting that can lead to more informed decisions on what parts of the system will benefit from SOA technologies [13].

- Architecture trade-off analysis —It is well known that trade-offs must be made in systems, because the accomplishment of a certain quality is often at the expense of another quality. Common examples of trade-offs are performance versus modifiability, availability versus safety, security versus performance, and interoperability versus cost.14 The use of service orientation in systems that have high system-quality requirements will require architectural trade-offs at the expense of loose coupling and flexibility. If the added overhead for a service-oriented system to meet quality requirements comes at the expense of the characteristics for which SOA is known, the decision to use service- oriented concepts should be reevaluated. An architecture analysis and evaluation method that is guided by business drivers and performed via scenarios in which the usage of SOA technologies is key can also help an architect make better, early, and informed decisions.

Finally, as service-oriented systems start to cross organizational boundaries, architects will have to reevaluate the use of SOA as an architectural style in these systems or to architect their systems in such a way that qualities are met without having to sacrifice the characteristics that have made SOA a worthwhile technology to adopt.

References

[1] This term was coined by Geoffrey A. Moore in his book Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers (Rev. ed. New York: Collins Business Essentials, 2006) and refers to the chasm that exists between visionaries (early adopters) and pragmatists (early majority) from a technology-adoption perspective.
[2] Softwareag.com. Software AG, Summer 2008. [Accessed July 13, 2009.]
[3] Forrester.com. Forrester, February 2009.
[4] Manadhata, Pratyusa K., Kamie M. C. Tan, Roy A. Maxion, and Jeannette M. Wing. CMU Technical Report CMU-CS-07-146, August 2007.
[5] Pardo-Castellote, Gerardo. SOA World Magazine, November 2007. [Accessed July 13, 2009.]
[6] Joshi, Rajive. Real-Time Innovations, Inc., August 2007.
[7] Simanta, Soumya, Ed Morris, Grace A. Lewis, Sriram Balasubramaniam, and Dennis B. Smith. Software Engineering Institute, June 2009. [Accessed July 13, 2009.]
[8] Kontogiannnis, Kostas, Grace A. Lewis, and Dennis B. Smith. “A Proposed Taxonomy for SOA Research.” In FSOA 2007). Software Engineering Institute, June 2008. [Accessed July 13, 2009.]
[9] Brun, Yuriy, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè and Mary Shaw. Lecture Notes in Computer Science Hot Topics, Volume 5525, 2009.
[10] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1994.
[11] Erl, Thomas. SOA Design Patterns. Upper Saddle River, NJ: Prentice Hall, 2009.
[12] Balasubramaniam, Sriram, Soumya Simanta, Ed Morris, Grace A. Lewis, and Dennis B. Smith. “Identity Management and its Impact on Federation in a System of Systems Context.” Proceedings of the 2009 3rd Annual IEEE Systems Conference, 2009.
[13] Lewis, Grace A., and Lutz Wrage. Software Engineering Institute, June 2005. [Accessed July 13, 2009.]
[14] Clements, Paul, Rick Kazman, and Mark Klein. Evaluating Software Architectures: Methods and Case Studies. Boston, MA; London: Addison-Wesley, 2001.

Grace Lewis is a Senior Member of the Technical Staff at the Software Engineering Institute (SEI) in Pittsburgh, PA. Currently, she is the lead for the System of Systems Engineering team within the Systems of Systems Practice (SoSP) initiative in the Research, Technology, and Systems Solutions (RTSS) program. Her current

interests and projects are in service-oriented architecture (SOA), technologies for systems interoperability, characterization of software-development life-cycle activities in systems of systems environments, and establishing an SOA research agenda. Grace's latest publications include multiple reports and articles on these subjects, as well as a book in the SEI Series in Software Engineering. She is also a member of the technical faculty for the Master in Software Engineering program at Carnegie Mellon University (CMU). Grace holds a B.Sc. in Systems Engineering; an Executive MBA from Icesi University in Cali, Colombia; and a Masters in Software Engineering from CMU.