Formalizing Service-Oriented Architectures

Khalil A. Abuosba and Asim A. El-Sheikh, The Arab Academy for Banking and Financial Sciences

Functional definitions of service-oriented architecture exist, but they generally lack explicit specification attributes, such as ownership and encapsulation mechanisms. Using Unified Modeling Language and Z-notation specification language, the authors provide a general-purpose, formal definition of SOA that helps clarify some ambiguities surrounding the architecture.

Service-oriented architecture (SOA) is a popular distributed transaction scheme for achieving interoperability in heterogeneous distributed systems deployments. Unfortunately, for all its popularity, SOA has a reputation of being ambiguous and misunderstood. The architecture’s many definitions only exacerbate the problem (see the “Related Work in Defining Service-Oriented Architectures” sidebar).

Addressing SOA as a highly abstracted mechanism untied to a specific implementation or technology eliminates these ambiguities. SOA can be viewed as an interaction between a service requester and a service provider. In this article, we address SOA from a logical and functional, rather than physical, perspective. Our abstract view of SOA is based on the minimum functional requirements expected for a transactional mechanism. We define these requirements in the Z-notation specification language, a tool for eliminating possible misunderstandings in natural languages.

W3C and OASIS Definitions

As the Internet’s largest subset, the World Wide Web is considered to be a global, SOA-based distributed information system. According to the World Wide Web Consortium (W3C), SOA is “a methodology for achieving application interoperability and reuse of IT assets that feature a strong architectural focus, an ideal level
Service-oriented architecture (SOA) can have different meanings to different people. For example, Steve Burbeck’s informal definition of SOA views the architecture as collaborations of the service provider, service requester, and service broker at runtime to achieve business processes.1 According to Eric Newcomer and Greg Lomow, “A [service-oriented] architecture is a style of design that guides all aspects of creating and using business services throughout their life cycle (from conception to retirement).”2 They continue, a SOA “is also a way to define and provision an IT infrastructure to allow different applications to exchange data and participate in business processes, regardless of their operating systems or programming languages underlying those applications.”

Dirk Krafzig, Karl Banke, and Dirk Slama define SOA as “a software architecture that is based on the key concepts of an application front end, service, service repository, and service bus. A service consists of a control, one or more interfaces, and an implementation.”3 Researchers also have attempted to define specific SOA implementations, such as Web services, but none have specified a general formal standardized SOA definition.

References

Visibility refers to the provider’s and requester’s ability to interact and accomplish the requested service capability. This dimension introduces opportunities for matching the requester’s needs to the provider’s capabilities and vice versa (see Figure 1). We can decompose visibility according to related attributes or functional requirements, such as awareness, willingness, reachability, and self-serviceability (see Figure 2).

Interaction—when the requester uses one of the provider’s capabilities—is mediated through message exchange and produces an effect (or set of effects). An effect can take the form of information resulting from using a capability, a state change in entities (defined or undefined), or a combination of both.2 Here, we define a service as a mechanism that lets a requester (consumer) use predefined capabilities.
A service interface specifies how to access a service’s capabilities. For a service to achieve visibility, the service interface of its information and behavior models must be exposed to requesters.

**Formal Definition in Z**

Z is a formal specification language developed by the Programming Research Group at Oxford University Computing Laboratory. A software specification language based on set theory and mathematical logic methods, Z notation reduces software development cost while enhancing software quality and reliability. Z notation is intended for functional properties only. It isn’t intended for nonfunctional properties, such as usability, performance, size, and reliability, nor for describing timed and concurrent behavior. Z notation’s main elements are:

- set theory and its components (standard set operators: union, intersection, and difference; set comprehension; Cartesian products; and power sets),
- mathematical logic (first-order predicate calculus),
- natural language,
- syntax notations (a set of symbols), and
- schema operators (such as negation, conjunction, and quantification).

To create a formal definition of SOA, we introduced a simple SOA abstraction using three Z-schema definitions based on the OASIS reference model. Our formal definition (see Figure 3) decomposes SOA as a consumer agent with needs, interacting with a provider agent that can generate effects. This definition provides a foundation for SOA environment development guideline decomposition.

We maintain a high abstraction level of this definition to provide a simple, general understanding of SOA environments. Using the Z/EVES (combined Z and Euclid Verification and Evaluation System) Mathematical Toolkit, we syntactically validated and proofed our definition schemas (see Figure 4).

**Goals, Properties, and Characteristics**

SOA is based on the notion of building a software solution for service providers and requesters to interact in heterogeneous environments. SOA aims to achieve distributed information systems’ level of transparency, which the architecture can realize fully or partially based on the system’s design complexity.

Why use SOA? The architecture can help developers achieve design goals, including reusability, interoperability, self-manageability, and high-level abstraction. SOA conformance properties include service self-containment, self-serviceability, dynamic location and invocation of services, network-accessible services, and published ser-
vice functionalities and modes of operation. As Roy Fielding wrote, “Architecture is often developed in the context of a predefined environment, such as the protocols, profiles, specifications, and standards that are pertinent.” Even though SOA isn’t a specification (it is considered to be an architectural style with characteristics and properties), we foresee Z notation as a formal utility that can direct SOA developers to appropriate guidelines while implementing their solutions.

**SOA Instances**

Web Services are a special implementation case of SOA that achieves most of the architecture’s properties by deploying decoupling mechanisms—specifically, loose coupling. Loose coupling is a primary functional requirement of Web service implementations, because a system implemented in a tight-coupling environment might not be functional or assure interoperability of services under different owners. Coupling concerns deployment-based factors in services, including interfaces, technologies, and processes.

For SOA deployment to be correct and complete, differentiating between functional and nonfunctional requirements is crucial. In our SOA interaction decomposition model, we abstracted the service interaction as a consumer agent, a provider agent, and an implementation. Here, we emphasize that SOA infrastructure’s functional requirements are made up of self-serviceability and dynamic discovery. SOA attributes abstract to a common payload protocol, published and discoverable interfaces, decoupling methods, and multiple communication interfaces and composability functions.

Some in the IT industry mistakenly refer to SOA as Web services. In contrast, Web services are actually instances or implementations of SOA. Other instances include technology-dependent SOA deployments, such as Distributed Component Object Model (DCOM) and the Common Object Request Broker Architecture (CORBA). DCOM implementations are platform-dependent, however, so they’ve had limited success. Any distributed systems architecture that meets the SOA conformance properties is an instance of SOA.

**Deployment Advantages**

SOA is a distributed business processes fulfillment methodology that focuses on business process engineering and execution. The architecture offers the advantage of reusing resources and executing tasks by composing services at runtime based on specified parameters. As long as the services conform to the SOA properties, developers can locate a service anywhere and fully implement location transparency (for example, site location concealment on the Web). Unlike a distributed object architecture, some instances of SOA (such as Web services) have characteristics that enable any service provider inside or outside an organization to offer services.

Another implicit and observable software engineering advantage of SOA is negotiation transparency. With this transparency, the service provider and the service requester needn’t negotiate about what the service does before the requester can incorporate it in an application. Applications can delay binding services until deployment or execution.

SOA is also cost effective. The requester can pay for services on the basis of use rather than on simply having access to the application. Implementing exception-handling schemes (an advantage for embedded applications) can further reduce costs.

Applications in SOA can be adaptive and reactive as needed. Moreover, SOA provides means for service interoperability in networks in which heterogeneity is a major obstacle. The
architecture also enables developers to construct new services on-demand through dynamic service construction at runtime. Overall, SOA is an extremely flexible and extensible architecture.

As a testament to its appeal, the architecture has attracted many private organizations as well as government agencies worldwide. The Canadian government values SOAs as architectures that “facilitate the manageable growth of large-scale enterprise systems, provide a simple scalable paradigm for organizing large networks of systems that require interoperability, minimize trust assumptions among providers and consumers to further promote greater business agility and autonomy, and still integrate functionality across ownership boundaries.”

**Drawbacks and Challenges**

Clearly, SOA has a lengthy list of benefits, but the architecture is not without drawbacks and challenges. Because SOA is a methodology, rather than a specification, SOA properties must exist to classify systems as SOA-based solutions. SOA reflects an abstraction of systems properties; what the architecture does and doesn’t address is neither defined nor specified clearly in the industry. However, the OASIS reference model differentiates between these aspects in an abstracted form.

We do know that SOA doesn’t address requirements such as trust, business transactions, authority, and delegation; additional infrastructures are necessary to resolve these issues. For the purpose of integration, services reference these infrastructures in the service descriptions and service interfaces.

One major challenge of deployment is that SOA instances require a management infrastructure of their own to achieve self-manageability. Additionally, on the efficiency side, processing SOA implementations based on XML technologies consumes a high percentage of CPU time.

Our formal, logical definition for SOA using the Z-notation specification language provides a clear, easily understandable meaning of SOA. In future work, we’ll further decompose our definition, perhaps including a detailed composition of constraints. We’ll also address formal SOA interactions specifications based on specific instances and implementations of SOA such as Web services.

**References**


**Khalil A. Abuosba**, formerly of the Arab Academy for Banking and Financial Sciences, is an assistant professor at Philadelphia University. His research interests include distributed systems architectures, Web security, and e-management. Abuosba has a PhD in computer information systems from the Faculty of Information Systems and Technology at the Arab Academy for Banking and Financial Sciences. Contact him at kh_abuosba@philadelphia.edu.jo.

**Asim A. El-Sheikh** is dean of Faculty of Information Systems and Technology at the Arab Academy for Banking and Financial Sciences. His research interests include software engineering and simulation. He has a PhD in software engineering from the University of London. Contact him at a.elsheikh@aabfs.org.