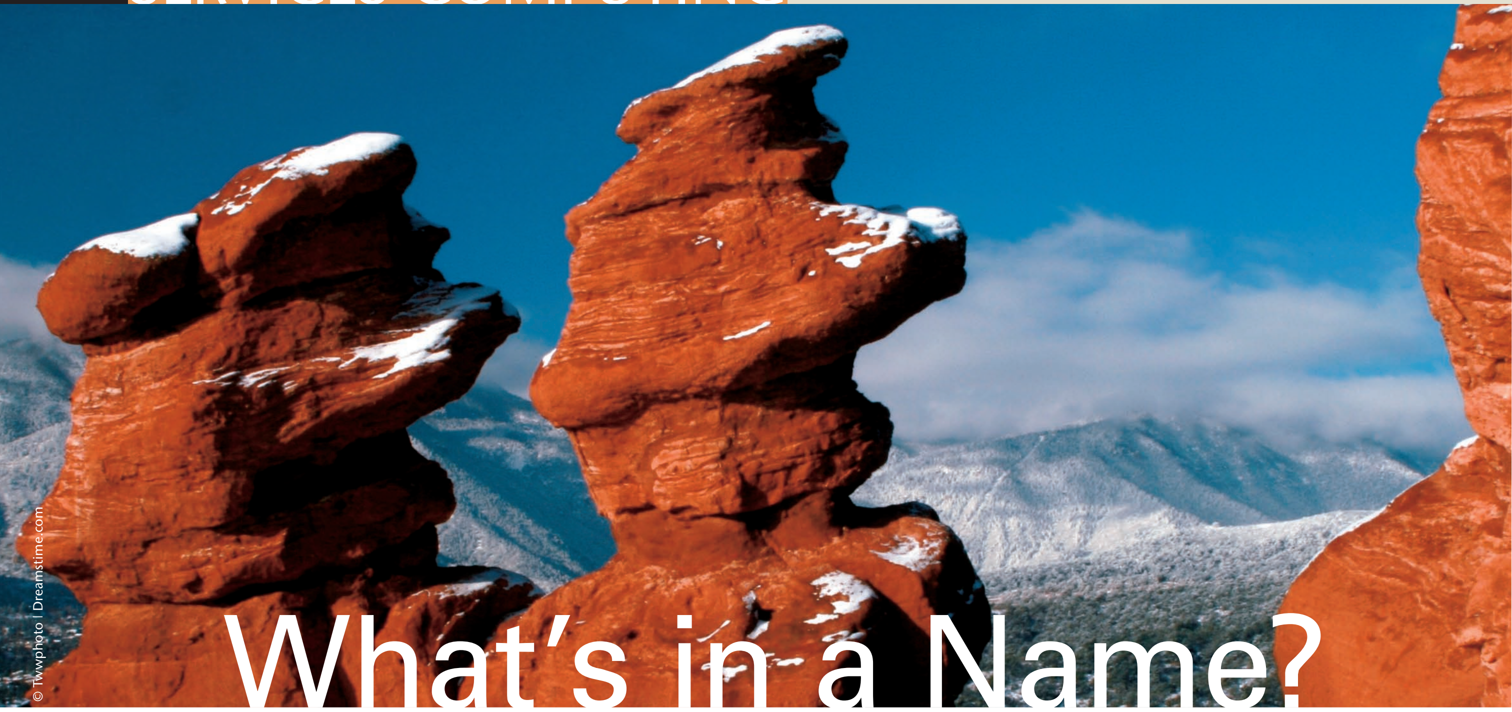

What's in a Name? Distinguishing between SaaS and SOA

Phillip A. Laplante, *Penn State University*
Jia Zhang, *Northern Illinois University*
Jeffrey Voas, *SAIC*

Considerable confusion arises in distinguishing between software as a service (SaaS) and service-oriented architecture (SOA). Zachman's framework can help to try to make sense of the alphabet soup of Web services and utilities that form the basis for both SOA and SaaS.

Various IT professionals have, at one time or another, erroneously used the terms software as a service (SaaS) and service-oriented architecture (SOA) interchangeably. At best, this faulty practice creates confusion; at its worst, it can lead to poor designs. Our goal, therefore, is to clarify the meaning of these two often used and abused terms.

Briefly stated, the difference between SaaS and SOA is that the former is a software-delivery model whereas the latter is a software-construction model. A better way to illuminate the differences between these two concepts is to use the well-known Zachman architectural model.[1]

In this article, we briefly examine the concepts of SaaS and SOA, followed by a brief history of software architectural models. We use the Zachman model to differentiate the two architectural approaches to building software. Because the Zachman model is so intuitive, the approach we take to describe the differences between SaaS and SOA works well even with non-IT professionals.

Defining the Terms

Sometimes known as *subscription software*,[2] the SaaS delivery model essentially separates software ownership from the user—the owner is a vendor who hosts the software and lets the user execute it on-demand through some form of client-side architecture via the Internet or an intranet. This new model delivers software as utility services and charges on a per-use basis, similar to the way a utility company charges for electricity. Perhaps the most celebrated SaaS product is the Salesforce.com tool for customer-relationship management. Yet, SaaS products are available for a wide range of business functions, including customer service, human resource management, desktop functionality, email, payroll, financial applications, and supply chain and inventory control.[3]

In a SOA model, the constituent components of the software system are reusable services.[4] A collection of services interact with each other through standard interfaces and communication protocols. SOA promises to fundamentally change the way we build internal systems as well as the way internal and external systems interact. This architectural strategy goes hand in glove with software applications that are close to business objects that help to create an abstraction layer (because SOA lets you select custom software "parts" that can align closely with their corresponding business functionality). SOA is also a consistent framework for plugging in appropriate software statically and dynamically.

Some of the major SOA players and their latest products include BEA AquaLogic, Sonic SOA Suite 6.1, Oracle Web Services Manager, HP Systinet Registry 6.0, Iona Artix 5.0, Cape Clear 7.5, Microsoft .NET, Sun Java Composite Application Platform Suite, and IBM WebSphere. In this list, you end up with a technology architecture, a process architecture, an application architecture, and so on. SOA helps bring these together, but it's not always easy to move in that direction with so many, diverse applications involved.

Despite their significant differences, SaaS and SOA are closely related architectural models for large-scale information systems. Using SaaS, a vendor can deliver a software system as a service. Using SOA enables the published service to be discovered and adopted as a service component to construct new software systems, which can also be published and delivered as new services. In other words, the two models complement each other: SaaS helps to offer components for SOA to use, and SOA helps to quickly realize SaaS.

Although both provide promising features for the modern software industry, they're just conceptual-level models and require detailed technology to support them. At present, the best-known enabler supporting both SaaS and SOA is Web services technologies—programmable Web applications with standard interface descriptions that provide universal accessibility through standard communication protocols.[5] Web services provide a holistic set of XML-based, ad hoc, industry-standard languages and protocols to support Web services descriptions (using Web Services Description Language [WSDL][6]), publication and discovery (using UDDI[7]), transportation (using SOAP[8]), and so on.

Neither SaaS nor SOA requires Web services technology, but it's by far the best current option for supporting them.

In other words, Web services technologies, with an associated stack of standards, enable and facilitate SaaS and SOA. It's worth noting that neither SaaS nor SOA requires Web services technology, but it's by far the best current option for supporting them. Given this fact, we use the terms *services* and *Web services* interchangeably throughout this article.

Software Architectures

Edsger Dijkstra first stressed that how software is partitioned and structured is important, and he introduced the idea of layered structures for operating systems.[9] The potential benefit of such a structure was to ease development and maintenance, but in a practical sense Dijkstra was laying the groundwork for modern operating systems design. David Parnas proposed several principles of software design[10] (which we would now view as architecture) that became the building blocks for modern software engineering:

- information hiding as the basis of decomposition for ease of maintenance and reuse;
- the separation of interface from component implementation;
- the `uses` relationship for controlling connectivity among components;
- the principles for error-detection and handling, identifying commonalities in "families of systems"; and
- the recognition that structure influences nonfunctional qualities of systems.

Table 1. Zachman's set of architectural models from different stakeholders' perspectives.

Stakeholder Perspective	Data	Function	Network
Objective/scope	List of entities important to the business	List of processes the business performs	Locations in which the business operates
Business model	Representation of business entities and rules	Representation of business resources and processes	Logistical representation of business units
Information system model	Requirement specification of data and objects	Requirements specification of interaction between data and objects	Software or system architecture
Technology model	Design specification of data and objects	Design specification for interaction among data and objects	Hardware and software components
Detailed representation	Database descriptions	Code	Network architecture
Functioning system	Data and objects	Function or interaction	Communications

Seminal work by Dewayne Perry and Alexander Wolf[11] introduced a model of software architecture that consisted of three components:

- *elements* included processing, data, and connecting elements;
- *form* defined the choice of architectural elements, their placement, and how they interact; and
- *rationale* defined the motivations for the choice of elements and form.

Barry Boehm later added the notion of *constraints* to the vision of software design to represent the conditions under which systems would produce win–lose or lose–lose outcomes for some stakeholders.[12] David Garlan and Mary Shaw provided an early introduction to various software architectural models and styles and how to use them together to facilitate software design.[13] In contrast with these works, which focused on single software applications, John Zachman examined architectures for large-scale information systems that encompass collections of communicating software applications[1]—the setting for both SOA and SaaS.

Zachman was the first to use a matrix framework for discussing an architecture in the context of information systems. As Table 1 (which we adapted from the original Zachman article[1]) shows, he believed that a comprehensive information system required a set of architectural models that represent different stakeholders' perspectives:

- An information system's *objective* or *scope* represents a ballpark view of the system (via user stories or use cases, for example).
- The *business model* is the owner's representation—often generated through traditional process mapping.
- The *information system model* is the designer's representation, which can take one of several architectural forms.
- The *technology model* is the builder's representation of the system.
- The *detailed representation* is an out-of-context representation of the system (looking at the software system without regard for its business purpose).
- Finally, there is the *functioning system* itself.

As Table 1 shows, representations of each of these views differ according to the dimensions of data, function, and network, because the connections between the components are via a network.

For our purposes, we narrowed the focus to just those cells in the Zachman model[1] that are of interest in comparing SOA and SaaS. As discussed earlier, both concentrate on connections among constituent components at large. They therefore belong to the *network* dimension. Using the corresponding cells (the far-right column in Table 1) let's focus on the differences between SOA and SaaS (see Table 2).

From the objectives/scope perspective, the SOA network model is a list of potential services to be used in a software system being built; the SaaS network model is a list of possible services to be delivered. From an owner's perspective, SOA implies a list of found business services to be used in the system; SaaS implies a list of business services to be provided. Using existing business

Table 2. Focused Zachman model for comparing software-oriented architecture (SOA) and software as a service (SaaS).

Stakeholder Perspective	Network (SOA)	Network (SaaS)
Objective/scope	List of possible services to use	List of possible services to deliver
Business model	List of business services to use	List of business services to provide
Information system model	Service component interaction model	Component interaction model
Technology model	Technology-dependent and platform-dependent service component interaction model	Technology-dependent and platform-dependent component interaction model
Detailed representation	List of technology-dependent languages and protocols used (such as UDDI, SOAP, XML, WSDL) and actual services used	Publish–subscribe architecture and notification facilities; list of technology-dependent languages, protocols, and services used (if any)
Functioning system	Interservice communication, coordination, and collaboration	Intercomponent communication, coordination, and collaboration

services could significantly eliminate software design and development expenses.

Note that SaaS doesn't mean that a software system is delivered as only one service. Instead, a software system could be delivered as multiple services—that is, parts of the system could be stand-alone services that work with the big service for the entire system.

From a designer's perspective, SOA depicts an architectural model describing interaction patterns among constituent service components, whereas SaaS describes interaction patterns among constituent components that aren't necessarily services. From a builder's perspective, both SOA and SaaS need to identify a technology (such as Web services) to realize the interaction models defined in the information system model.

The list of detailed languages and protocols must also be identified—for example, WSDL for description, UDDI for publishing, and SOAP for communication. Meanwhile, both SOA and SaaS must consider platform-dependent designs. For a SOA-based software construction, the developer must choose a platform to carry the Web services technology—for example, whether to go with BEA AquaLogic, IBM WebSphere, or Microsoft .NET. For SaaS, the developer must also decide which platform to use in implementing the services.

Developers will implement invocations to actual services as part of the detailed representation. Functioning SOA-based systems require monitoring and management of all communication, coordination, and collaboration among service components. A functioning SaaS requires management of the communication, coordination, and collaboration among its internal components.

By keeping SOA in mind while creating a SaaS, developers can intentionally produce multiple services at various granular levels. In this way, more services at various complexity levels can become available and thus facilitate more SOA-based construction.

SaaS and SOA are important emerging technologies that are gaining wider entry into business. Nonetheless, both are sometimes misunderstood. Using the traditional Zachman model to describe the nature of these two important technologies can help enlighten architectural choices and aid designers and developers in preparing appropriate designs and implementations. Finally, we hope using the Zachman model to describe these technologies will be helpful in educating non-IT professionals. IT

References

1. J.A. Zachman, "A Framework for Information Systems Architecture," *IBM Systems J.*, 1987, vol. 26, no. 3, pp. 276–292.
2. M. Turner, D. Budgen, and P. Brereton, "Turning Software into a Service," *Computer*, vol. 36, no. 10, 2003, pp. 38–44.
3. M.H. Weier and L. Smith, "Businesses Get Serious about Software as a Service," *Information Week*, 16 Apr. 2007, pp. 46–48.

4. L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*, Springer, 2007.
5. C. Ferris and J. Farrell, "What Are Web Services?" *Comm. ACM*, vol. 46, no. 6, 2003, p. 31.
6. "Web Services Description Language (WSDL) 1.1," W3C note, E. Christensen et al., eds., 15 Mar. 2001; www.w3.org/TR/wsdl.
7. *Universal Description, Discovery, and Integration (UDDI), version 3*, Organization for the Advancement of Structured Information Standards (Oasis), 2004; www.uddi.org/pubs/uddi_v3.htm.
8. *SOAP Version 1.2, Part 1: Messaging Framework (second edition)*, W3C recommendation, M. Gudgin et al., eds., 27 Apr. 2007; www.w3.org/TR/soap12-part1/.
9. E. Dijkstra, "The Structure of the 't.h.e.' Multiprogramming System," *Comm. ACM*, vol. 18, no. 8, 1968, pp. 453–457.
10. D.L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Comm. ACM*, vol. 15, no. 12, 1972, pp. 1053–1058.
11. D. Perry and A. Wolf, "Foundations for the Study of Software Architecture," *ACM Sigsoft Software Eng. Notes*, vol. 17, no. 4, 1992, pp. 40–52.
12. B. Boehm, "Anchoring the Software Process," *IEEE Software*, vol. 13, no. 4, 1996, pp. 73–82.
13. D. Garlan and M. Shaw, "An Introduction to Software Architecture," *Advances in Software Eng. and Knowledge Eng., vol. 2*, V. Ambriola and G. Tortora, eds., World Scientific Publishing, 1993, pp. 1–39.

Phillip A. Laplante *is a professor of software engineering at Penn State University and serves as the CTO for the Eastern Technology Council. He is a fellow of the IEEE and SPIE—the Optical Society. Contact him at plaplante@psu.edu.*

Jia Zhang *is an assistant professor in the Department of Computer Science at Northern Illinois University. She is a member of the IEEE.*

Jeffrey Voas *is the director of systems assurance and a technical fellow at SAIC. He is a senior member of IEEE.*