RAMOS LUTHER

0912200546

Service Oriented Architecture

Definitions

In computing, service oriented architecture (SOA) provides methods for systems development and integration where systems group functionality around business processes and package these as *interoperable services*. A SOA infrastructure allows different applications to exchange data with one another as they participate in business processes. Service-orientation aims at a *loose coupling* of services with operating systems, programming languages and other technologies that underlie application. SOA separates functions into distinct units, or service, which developers make accessible over a network in order that users can combine and reuse them in the production of business application. These services communicate with each other by passing data from one service to another, or by coordinating an activity between two or more services. Many commentators see SOA concepts as built upon and evolving from older concepts of distributed computing and modular programming.

Companies have long sought to integrate existing systems in order to implement information technology (IT) support for business processes that cover all present and prospective systems requirements needed to run the business end-to-end. A variety of designs serve this end, ranging from rigid point-to-point electronic data interchange (EDI) interactions to web auctions. By updating older technologies, for example by Internet-enabling EDI-based systems, companies can make their IT systems available to internal or external customers; but the resulting systems have not proven flexible enough to meet business demands that require a flexible, standardized architecture to better support the connection of various applications and the sharing of data.

SOA offers one such prospective architecture. It unifies business processes by structuring large applications as an *ad hoc* collection of smaller modules called "services". Different groups of people both inside and outside an organization can use these applications, and new applications built from a mix of services from the global pool exhibit greater flexibility and uniformity. One should not, for example, have to provide redundantly the same personal information to open an online checking, savings or IRA account, and further, the interfaces one interacts with should have the same look and feel and use the same level and type of input-data validation. Building all applications from the same pool of services makes achieving this goal much easier and more deployable to affiliated companies. For example: interacting with a rental-car company's reservation system from an airline's reservation system.

Service Oriented Architecture (SOA) provides a design framework with a view to realizing rapid and low-cost system development and to improving total system-quality. SOA may use web services standards and web technologies and is rapidly becoming a standard approach for enterprise information systems.

Web services face significant challenges because of particular requirements. Applying the SOA paradigm to a real-time system presents many problems, including response time, support of event-driven, asynchronous parallel applications, complicated human interface support, reliability, etc. This article defines SOA and includes detailed discussion on several issues that arise when applying SOA to industrial systems.

One can define a *service-oriented architecture* (SOA) as a group of services that communicate with each other. The process of communication involves either simple data-passing or two or more services coordinating some activity. Intercommunication implies the need for some means of connecting two or more services to each other.

SOAs build applications out of software services. Services comprise intrinsically unassociated units of functionality that have no calls to each other embedded in them. They typically implement functionality most humans would recognize as a service, such as filling out an online application for an account, viewing an online bank-statement, or placing an online booking or airline ticket order. Instead of services embedding calls to each other in their source code, they use defined protocols that describe how one or more services can "talk" to each other. This architecture then relies on a business process expert to link and sequence services, in a process known as orchestration, to meet a new or existing business system requirement.

Relative to typical practices of earlier attempts to promote software reuse via modularity of functions or by use of predefined groups of functions known as classes, SOA's atomic-level objects often end up 100 to 1,000 times larger.

A software developer or software engineer associates individual SOA objects by using orchestration. In the process of orchestration, a software engineer or process engineer associates relatively large chunks of software functionality (services) in a non-hierarchical arrangement (in contrast to a class hierarchy) by using a special software tool that contains an exhaustive list of all of the services, their characteristics, and a means to record the designer's choices that the designer can manage and the software system can consume and use at run-time.

Underlying and enabling all of this requires metadata in sufficient detail to describe not only the characteristics of these services, but also the data that drives them. Programmers have made extensive use of XML in SOA to structure data that they wrap in a nearly exhaustive description-container. Analogously, WSDL typically describe the services themselves, while SOAP describes the communications protocols. Whether these description languages are the best possible for the job, and whether they will remain the favorites in the future, remains an open question. In the meantime SOA depends on data and services that are described using some implementation of metadata that meets the following two criteria:

1. the metadata must come in a form that software systems can use to configure dynamically by discovery and incorporation of defined services, and also to maintain coherence and integrity
2. the metadata must also come in a form that system designers can understand and manage with a reasonable expenditure of cost and effort

SOA has the goal of allowing users to string together fairly large chunks of functionality to form *ad hoc* applications that are built almost entirely from existing software services. The larger the chunks, the fewer the interface points required implementing any given set of functionality; however, very large chunks of functionality may not prove sufficiently granular for easy reuse.

SOA is a design for linking computational resources (principally applications and data) on demand to achieve the desired results for service consumers (either end users or other services). OASIS (the Organization for the Advancement of Structured Information Standards) defines SOA as the following:

Benefits

Enterprise architects believe that SOA can help businesses respond more quickly and cost-effectively to changing market conditions. This style of architecture promotes reuse at the macro (service) level rather than micro (classes) level. It can also simplify interconnection to – and usage of – existing IT (legacy) assets.

In some respects, one can regard SOA as an architectural evolution rather than as a revolution. It captures many of the best practices of previous software architectures. In communications systems, for example, little development has taken place of solutions that use truly static bindings to talk to other equipment in the network. By formally embracing a SOA approach, such systems are better positioned to stress the importance of well-defined, highly inter-operable interfaces.

Some have questioned whether SOA is just a revival of modular programming (1970s), event-oriented design (1980s) or interface/component-based design (1990s). SOA promotes the goal of separating users (consumers) from the service implementations. Services can therefore be run on various distributed platforms and be accessed across networks. This can also maximize reuse of services.

SOA is an architectural and design discipline conceived to achieve the goals of increased interoperability (information exchange, reusability, and compos ability), increased federation (uniting resources and applications while maintaining their individual autonomy and self-governance), and increased business and technology domain alignment.

Service-Oriented Architecture (SOA) is an architectural approach (or style) for constructing complex software-intensive systems from a set of universally interconnected and interdependent building blocks, called services.

SOA realizes its business and IT benefits through utilizing an analysis and design methodology when creating services that ensures they are consistent with the architectural vision and roadmap, and adhere to principles of service-orientation. Arguments supporting the business and management aspects from SOA are outlined in various publications.

A service comprises a stand-alone unit of functionality available only via a formally defined interface. Services can be some kind of "nano-enterprises" that are easy to produce and improve. Also services can be "mega-corporations" that are constructed as coordinated work of sub-ordinate services.

A mature rollout of SOA effectively defines the API of the organization.

Implementation of services should be treated as separate projects from the larger project for three reasons:

- It promotes the concept to the business that services can be delivered quickly and independently from the larger and slower-moving projects common in the organization. The business starts understanding systems and simplified user interfaces calling on services. This advocates agility.
- It promotes the decoupling of services from its consuming project. This encourages good design where the service is designed without knowing who its consumers are.
- Documentation and test artifacts of the service are not embedded within the detail of the larger project. This is important when the service needs to be reused later.

An indirect benefit of SOA is dramatically simplified testing. Services are autonomous, stateless, with fully documented interfaces, and separate from the cross-cutting concerns of the implementation. The industry has never been exposed to this circumstance before.

If appropriate test data is defined in the organization, then when a service is being built, a corresponding stub is built that reacts to the test data. A full set of regression tests, scripts, data, and responses is also captured for the service. The service can be tested as a 'black box' using existing stubs corresponding to the services it calls. Test environments can be constructed where the primitive and out-of-scope services are stubs, while the remainder of the mesh is test deployments of full services. As each interface is fully documented, with its own full set of regression test documentation, it becomes simple to identify problems in test services. Testing evolves to merely validating that the test service operates according to its documentation, and in finding gaps in documentation and test cases of all services within the environment. Managing data state of idempotent services is the only complexity.