
Abstract

SOA is relatively new, so companies seeking to implement it cannot tap into a wealth of practical expertise. Without a common language and industry vocabulary based on shared experience, SOA may end up adding more custom logic and increased complexity to IT infrastructure, instead of delivering on its promise of intra and inter-enterprise services reuse and process interoperability. To help develop a shared language and collective body of knowledge about SOA, a group of SOA practitioners created this SOA Practitioners' Guide series of documents. In it, these SOA experts describe and document best practices and key learnings relating to SOA, to help other companies address the challenges of SOA. The SOA Practitioners' Guide is envisioned as a multi-part collection of publications that can act as a standard reference encyclopedia for all SOA stakeholders.

1.1 Intended Audience

This document is intended for the following audience:

- Business and IT leaders, who need to start and manage an SOA strategy across the enterprise/LOB
- Enterprise Architects who need to drive the vision and roadmap of the SOA program and the architecture of each implementation that falls under it
- Program Managers who need to manage a portfolio of sub-projects within an overall SOA business strategy
- Project Team Members, who need to map dependencies and develop a timeline that meets the business expectations
- Vendors who provide solutions and tools for new business capabilities to the business and IT
- Standards bodies which need a better understanding of use cases of how business and IT plan to leverage technology to meet their objectives.

Services Lifecycle Stages

1.2 Composite Application Design

1.2.1 Actors

- Project manager (IT)
- Business analysts
- Enterprise architects
- Project architects
- Designers
- Technical leads or lead developer

1.2.2 Tools Used

- Design: Rational, Together Architecture, Eclipse, and others

1.2.3 Artifacts (Deliverables)

- Design models: UML, SCA service assembly model, and others
- Bindings: JMS, RMI, IIOP, HTTP(s), and others

1.2.3.1 Artifact Description

This stage of the service lifecycle generates models that represent the system flow, data flow, enterprise data model (represented in the form of Entity Relationship Diagram(ERD)), application design (represented in UML), activity diagram, and sequence diagram. During this phase, the team also generates the high-level deployment model, identifying the servers, OS, middleware, databases, firewall, and load balancers.

The application designer could be an architect, technical lead, or lead developer, and may decide to use some of the tools in the market. Typically this tool is based on RUP or a variation of RUP and would consist of artifacts such as activity diagrams, use cases, class diagrams, ERDs, deployment models, and deployment models. Architects should share these artifacts with the team for review and approval, and provide instructions to the development teams.

1.2.4 Service Lifecycle Stage Key Considerations

1.2.4.1 Enterprise Architecture Framework

IT organizations should standardize on an architecture framework that defines architecture standards, development processes, design patterns, and tools. Most IT organizations will already have adopted one of the standard application lifecycle management processes and modified it to fit its own needs. In addition, there are other well known architecture frameworks such as Zachman, Federal Enterprise Architecture (FEA) and The Open-Group Architecture Framework (TOGAF). IT must adopt an architecture framework enterprise-wide to be successful in implementing SOA.

1.2.4.2 Services Classification Framework

A services classification framework helps provide a basis for design and development of services and is essential to achieving a flexible architecture. Services could be classified into multiple categories such as:

- SOA reference architecture, including shared data service, business process, and portal service
- Portfolio of services, including quote-to-cash services and mortgage approval services
- LOB services, including sales and support services.

After the services have been identified, the team classifies them based on the defined standards. Classification helps business managers, project managers, and the development team to identify what services are being developed and for what purpose. This makes it easier for the project manager to distribute development tasks to the appropriate teams.

1.2.4.3 Service Granularity

Service granularity refers to the level of abstraction or how much functionality the service covers. Teams can apply the concept of granularity either to the service itself or to the service methods.

In determining service granularity, architects need to consider performance requirements. Fine-grained services such as Enterprise Java Beans (EJBs) are typically easier to understand and implement, since in many cases much of the work is already done. However, if performance is a consideration, then aligning services with existing EJBs may *not* turn out to be the optimal solution. A fine-grained architecture that relies on multiple request and response pairs (a “chatty” protocol) may offer slower performance. In order to reduce the effects of network latency, system I/O, and thread/process wait states, it is much better to create a coarse-grained service that internally composes multiple business domain services and uses fewer messages.

Service granularity must take into account legacy system interfaces that are (and must remain) unaware of the new protocol. Architects must plan carefully to avoid any changes to legacy systems when adding a new data channel in the form of the Web service.

Finally, in determining granularity architects must consider the possibility of future changes to the underlying implementation. In general, businesses benefit from using coarse-grained facades or patterns to hide the fine-grained services beneath them. The goal is to insulate services from changes to the underlying implementation by designing them at a level of granularity that will allow for future expansion with little impact to the clients.

1.2.4.4 Reuse Strategy

Reuse of a component starts at the design layer. Architects should design components so that the client use of that component only executes or inherits the methods that are needed to perform a given task. Shared components must be written as standalone elements that perform a task while hiding its complexity. Architects should consider using the facade pattern, as it tends to encapsulate each member of a related task or class within a common interface so that client code may use those tasks interchangeably. Since a task is usually coded as one or a few isolated methods, encapsulation emphasizes reuse of methods that perform a single task. Reusing a single task is easier than reusing objects containing code and data, which may perform multiple tasks.

1.2.5 Service Lifecycle Stage Recommended Process

This phase begins as soon as the project manager or business analyst hands over the requirements to the technical team. The team then begins composite application design, typically by following these steps::

1. Architect downloads requirements from SCM/SOA repository, reviews them, and submits them to business analyst for further clarifications, if required
2. Architect selects tools to model and design the application
3. Architect identifies services and may search the SOA repository to identify potential reuse
4. Architect defines services, implementations, bindings, and dependencies
5. Once the design is complete, architect uploads all design artifacts to the SOA repository for approval
6. Architecture review board—consisting of enterprise architects, project architects, business analysts, and the project manager—reviews design to ensure architecture meets business requirements and is consistent across the enterprise.

The SOA repository should provide capability to generate service templates for each service category based on patterns defined by the enterprise architects.

Contributing SOA Practitioners

Surekha Durvasula, Enterprise Architect, Kohls

Martin Guttman, Principal Architect, Customer Solutions Group, Intel Corp

Ashok Kumar, Manager, SOA Architecture, Avis/Budget

Jeffery Lamb, Enterprise Architect, Wells Fargo

Tom Mitchell, Lead Technical Architect, Wells Fargo Private Client Services

Burc Oral, Individual Contributor

Yogish Pai, Chief Architect AquaLogic Composer, BEA Systems, Inc.

Tom Sedlack, Enterprise Architecture & Engineering, SunTrust Banks, Inc.

Dr Harsh Sharma, Senior Information Architect, MetLife

Sankar Ram Sundaresan, Chief Architect e-Business, HP-IT