Design an SOA solution using a reference architecture

Improve your development process using the SOA solution stack

Ali Arsanjani, Ph.D. (arsanjan@us.ibm.com)
Chief Architect
IBM

Liang-Jie Zhang (zhanglj@us.ibm.com)
Research Staff Member
IBM

Michael Ellis (msellis@ca.ibm.com)
Solution Architect
IBM

Abdul Allam (allam@us.ibm.com)
Certified IT Architect
IBM

Kishore Channabasavaiah (kishorec@us.ibm.com)
Executive Architect
IBM

13 August 2014
(First published 28 March 2007)

IBM Service-Oriented Architecture experts define an SOA reference architecture based on multiple projects in various industries over the past three years. You can use the reference architecture to organize your own SOA design, development, and deployment to obtain better results and return on investment. The reference architecture, also referred to as the SOA solution stack, defines the layers, architectural building blocks, architectural and design decisions, patterns, options, and the separation of concerns that can help your enterprise better realize the value of SOA.
13 August 2014 - *Updated broken link in Resources: Ali Arsanjani's blog*

Overview

The SOA reference architecture provides a blueprint for "fractal scope," from an ecosystem architecture to an enterprise or application architecture and is based on establishing the building blocks of SOA: services, components, and flows that collectively support business processes

and goals. The metadata underlying each layer and the relationships between layers can further facilitate SOA in bridging the gap between business and IT from solution modeling to solution realization. The other major capability afforded by the SOA solution stack is the increase of reusability when designing and developing solution assets for rapid development, deployment, and management of SOA solutions within industry or across industries.

Introducing the SOA solution stack

Over the past 25 years software architecture has grown rapidly as a discipline. With the pioneering work of Shaw and Garlan on software architecture [7], and later additions to their ideas, we came to recognize the need for architectural patterns and styles. An architectural style is a set of components (which we will call "architectural building blocks," or ABBs, to emphasize their generic nature), connectors, constraints composition, containers, and configurations [7]. Dr. Ali Arsanjani refers to these as the "six Cs" of an architectural style [8].

In recent years, the decoupling of interface from implementation at the programming level has become a popular architectural approach because such decoupling facilitates the creation of complex systems [6] required by today's business applications. According to this approach, the interface of a service consumed by a service consumer is loosely coupled with its implementation by a service provider and the implementation is decoupled from its binding. This style has become the key characteristic of Service-Oriented Architecture; that is, rather than the implementations of components being exposed and known, only the services provided are published and consumers are insulated from the details of the underlying implementation by a provider or broker.

SOA allows business and IT convergence through agreement on a set of business-aligned IT services that collectively support an organization's business processes and goals. Not only does it provide flexible, decoupled functionality that can be reused, but it also provides the mechanisms to externalize variations of quality of service; for example, in declarative specifications such as WS-Policy and related standards [8].

As a flexible and extensible architectural framework, SOA has the following unique capabilities:

- **Reduced cost.** Provides the opportunity to consolidate redundant application functionality and decouple functionality from obsolete and increasingly costly applications while leveraging existing investments.
- **Increased flexibility.** Structures IT applications based on services in such a way as to facilitate the rapid restructuring and reconfiguration of the business processes and applications that consume them.
- **Increased revenue.** Provides the opportunity to enter into new markets and leverage existing business capabilities in new and innovative ways using a set of loosely coupled IT services. Helps to increase market share by offering new and better business services.
- **Added agility.** Delivers business-aligned applications faster.
- **Increased consolidation.** Integrates IT systems across siloed applications and organizations.

However, there are significant challenges in creating an SOA solution within the context of customer engagements. For example, from a technical perspective, the architect needs to answer questions such as:

- How do we produce an SOA solution using a well-defined notation?
- How can an SOA solution be organized as an architectural framework with interconnected architectures and transformation capabilities?
- How can an SOA solution be designed in a manner that maximizes asset reuse?
- How can automated tools take the guesswork out of architecture validation and capacity planning?

In order to address these issues, this article presents an SOA "solution stack" that includes a reference architecture (an architectural template, or blueprint) for a Service-Oriented Architecture. It provides a high-level abstraction of an SOA factored into layers, each of which addresses specific value propositions within SOA. Underlying this layered architecture is a metamodel consisting of layers, architectural building blocks (ABBs), relations between ABBs and layers, interaction patterns, options, and architectural decisions. These will guide the architect in the creation of the architecture.

The layers facilitate separation of concerns and assist the process of creating an SOA in conjunction with methods such as the Service-Oriented Modeling and Architecture (SOMA) method [1]. The SOA solution stack defines a blueprint that can be used to define the layers, architectural building blocks within the layers, options available at each layer, and typical architectural decisions that need to be made.

Layers of the SOA solution stack

The SOA solution stack has nine layers that are designed to reinforce SOA business value. For each layer, there are two aspects: logical and physical. The logical aspect includes all the architectural building blocks, design decisions, options, key performance indicators, and the like; the physical aspect of each layer covers the realization of each logical aspect using technology and products. This article will focus on the logical aspect of the SOA solution stack.

The logical aspect of the solution stack addresses the question, "If I build a SOA, what would it conceptually look like and what abstractions should be present?" The solution stack enumerates the fundamental elements of an SOA solution and provides the architectural foundation for the solution.

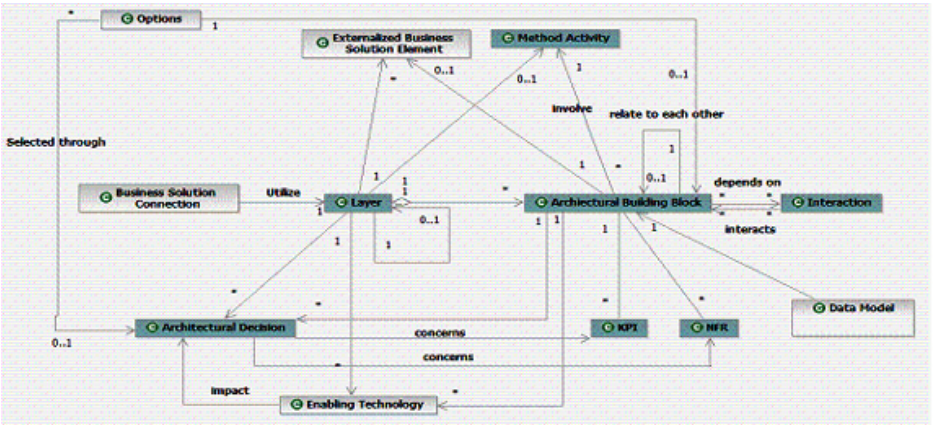
As shown in Figure 1, the metamodel of solution stack includes the following elements:

- **Layer.** An abstraction of the nine layers of the SOA solution stack that contains a set of characteristics, including architectural building blocks, architectural decisions, interactions among components, and interactions among layers.
- **Option.** A collection of possible options available in each layer that impacts other artifacts of a layer. Options are the basis for architectural decisions within and between layers.
- **Architectural decision.** A conclusion derived from options. The architectural decision involves architectural building blocks, key performance indicators, and nonfunctional

requirements to provide information on configuration and usage of architectural building blocks. Existing architectural decisions can also be reusable by some layers or architectural building blocks.

- **Method activity.** A collection of steps that involve architectural building blocks to form a process in a layer.
- **Architectural building blocks.** Reside in a layer and contain attributes, dependencies, and constraints as well as relationships with other architectural building blocks in the same layer or different layers.
- **Interaction pattern.** An abstraction of the various relationships among architectural building blocks; for example, patterns and diagrams.
- **Key performance indicator (KPI).** A constraint on architectural building blocks.
- **Nonfunctional requirement (NFR).** A constraint on architectural building blocks.
- **Enabling technology.** A technical realization of architectural building blocks in a specific layer.
- **Externalized business solution element.** A business service entity in a specific layer to be exposed to external consumers.
- **Business solution connection.** An adaptor for utilizing external services.
- **Data model.** Models data content associated with architectural building blocks, including data exchange between layers and external services.

Figure 1. A metamodel for instantiating the SOA reference architecture for a given solution



The architectural diagram shown in Figure 2 depicts an SOA as a set of logical layers. Note that the SOA solution stack is a partially layered architecture. One layer does not solely depend upon the layer below it; for example, a consumer can access the business process layer as a service or the service layer directly, but not beyond the constraints of the SOA architectural style. Further, a given SOA solution may exclude a business process layer and have a consumers layer that interacts directly with the services layer. Such a solution would not benefit from the business value associated with the business process layer; however, that value could be achieved at a later stage by adding the layer. The degree to which a given organization realizes the full SOA solution stack will differ according to the level of service integration maturity it requires.

Figure 2. Layers of the SOA reference architecture: Solution stack view

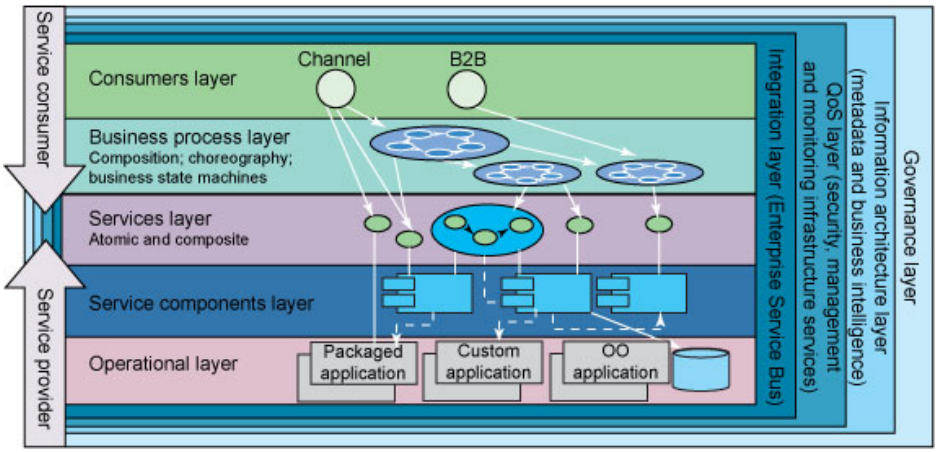


Figure 2 illustrates the multiple separations of concern in the nine layers of this reference architecture. Although the provider and the consumer can belong to the same organization -- and they usually do -- the SOA solution stack does not assume this to be the case. The main point of the provider/consumer separation is that there is value in decoupling one from the other along the lines of a business relationship. Organizations may have different lines of business that use this architectural template (where one organization is the consumer and another is the provider), customizing it for their own needs and integrating and interacting between organizations. In such cases there is still real value in maintaining a decoupled consumer/provider relationship. The lower layers (services, service components and operational layer) are concerns for the provider, and the upper ones (services, business processes, and consumers) are concerns for the consumer. In the remainder of this article we describe each layer and, in subsequent sections, describe the relationships between the layers.

Note that there are five horizontal layers that relate to the overall functionality of the SOA solution. The vertical layers are nonfunctional in nature and support various concerns that cut across the functional layers.

SOA solution stack assumptions

The SOA solution stack assumes the following:

- A set of requirements (service requirements) exists that collectively establishes the objective of the SOA. These requirements are both functional and nonfunctional in nature. Nonfunctional service aspects include security, availability, reliability, manageability, scalability, and latency.
- A service requirement is the documented capability that a service is expected to deliver. The provider view of a service requirement is the business and technical capability that a specific service needs to deliver given the context of all of its consumers. The consumer view of a service requirement is the business and technical capability that the service is expected to deliver in the context of that consumer alone.
- The fulfillment of any service requirement may be achieved through the capabilities of one layer or a combination of layers in the SOA solution stack.

- For each layer there is a specific mechanism by which the service requirements influence that layer.
- The identification of service requirements and the mapping of those requirements to each of the layers of the solution stack is a key aspect in developing an SOA for an enterprise.

Layer 1. Operational layer

This layer includes all custom or packaged application assets in the application portfolio running in an IT operating environment, supporting business activities.

The operational layer is made up of existing application software systems; thereby, it is used to leverage existing IT investments in implementing an SOA solution. This directly influences the overall cost of implementing the SOA solution, which can help free up budget for new initiatives and development of new business-critical services. A number of existing software systems are part of this layer. Those systems include:

- Existing monolithic custom applications, including J2EE™ and Microsoft® .NET® applications
- Legacy applications and systems
- Existing transaction processing systems
- Existing databases
- Existing packaged applications and solutions, including enterprise resource planning (ERP) and customer relationship management (CRM) packages (such as SAP and Oracle solutions)

Layer 2. Service component layer

This layer contains software components, each of which provide the implementation for, realization of, or operation on a service, which is why it's called a service component. Service components reflect the definition of a service, both in its functionality and its quality of service. Service components may comply with the Service Component Architecture (SCA) and Service Data Objects (SDO) specifications [3].

The service component layer conforms to service contracts defined in the services layer; it guarantees the alignment of IT implementation with service description.

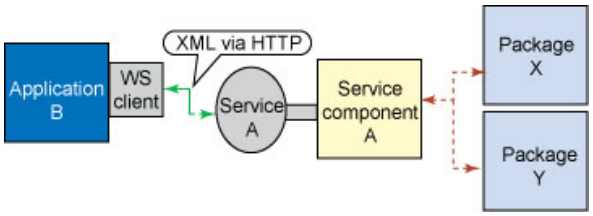
Each service component:

- Provides an enforcement point for "faithful" service realization to ensure quality of service and adherence to service-level agreements (SLAs).
- Enables business flexibility by supporting the functional implementation of IT flexible services as well as their composition and layering.
- Enables IT flexibility by strengthening decoupling in the system. Decoupling is achieved by hiding volatile implementation details from consumers.

Figure 3 illustrates these concepts and shows service A implemented using a combination of behavior from the third-party Package X and Application Y. Application B, the consumer, is coupled only to the description of the exposed service. The consumer must assume that the realization of the service is faithful to its published description (thereby providing service compliance), and it is the providers' responsibility to ensure that it is. The details of the realization, however, are of no consequence to Application B. Service Component A acts as a service implementation facade,

aggregating available system behavior and giving the provider an enforcement point for service compliance.

Figure 3. Service component as a facade



Subsequently, the provider organization may decide to replace Package X with Package M or some other application. Service component A encapsulates the required modifications with the result that there is no impact on any consumers of service A. This example illustrates the value of the service component layer in supporting IT flexibility through encapsulation.

Layer 3. Services layer

This layer consists of all the services defined within the SOA. For the purposes of this reference architecture, a service is considered to be an abstract specification of a collection of (one or more) business-aligned IT functions. The specification provides consumers with sufficient detail to invoke the business functions exposed by a provider of the service; ideally this is done in a platform-independent manner. The service specification includes a description of the abstract functionality offered by the service similar to the abstract stage of a Web Services Definition Language (WSDL) description [5]. This information is not necessarily written using WSDL.

The service specification may also include:

- A policy document
- SOA management descriptions
- Attachments that categorize or show service dependencies

Some of the services in the service layer may be versions of other services, implying that a significant successor-predecessor relationship exists between them.

Exposed services reside in this layer; they can be discovered and invoked or possibly choreographed to create a composite service. Services are functions that are accessible across a network through well-defined interfaces of the services layer. The service layer also takes enterprise-scale components, business-unit-specific components, and project-specific components and externalizes a subset of their interfaces in the form of service descriptions. Thus, the components provide services through their interfaces. The interfaces are exported as service descriptions in this layer, where services exist in isolation (atomic) or as composite services.

This layer contains the contracts (service descriptions) that bind the provider and consumer. Services are offered by service providers and are consumed by service consumers (service requestors).

Services and their underlying building blocks are defined according to the service identification activities defined through three complementary techniques:

- Domain decomposition
- Existing asset analysis
- Goal-service modeling

These techniques are described by the Service-Oriented Modeling and Architecture (SOMA) method for the identification, specification, and realization of services, components and flows [1]. They represent, therefore, the heart of the SOA value proposition -- improved agility from the decoupling of business and IT. The quality of these service definitions have a significant impact on the benefit of a given SOA effort. (See service litmus tests in the Specification phase of the SOMA method.)

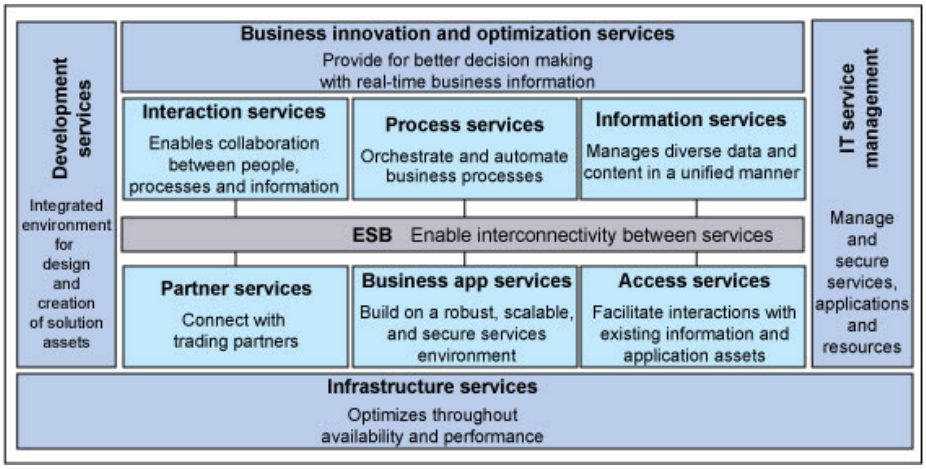
Services are accessible independent of implementation and transport. This capability allows a service to be exposed consistently across multiple customer-facing channels such as the Web, interactive voice response (IVR), Seibel client (used by a customer service rep), and so on. The transformation of responses to HTML (for Web), Voice XML (for IVR), XML string (for Siebel client) can be done through XSLT functionality supported through Enterprise Service Bus (ESB) transformation capability in the integration layer.

It is important to acknowledge that service components may consume services to support integration. The identification and exposure of this type of service (that is, internal services) does not necessarily require the same rigor as is required for a business service. While there may be a compelling IT-related reason behind the use of such services, they are not generally tied to a business process. As such, they do not warrant the rigorous analysis required for business services.

This set of requirements and services contained by this layer can be used to better leverage the various capabilities provided by a mix of different vendors. This is because the requirements enable the objective identification of SOA infrastructure requirements. The solution stack provides a well-factored decomposition of the SOA problem space, which allows architects to focus on those parts of an SOA solution that are important in the context of the problem they are solving and to map the required capabilities to vendor product capabilities. This is preferred to trying to reverse-engineer an SOA solution architecture from the capability of a particular vendor's products. So, in addition to being an important template for defining an SOA solution at a logical level, this layer of the SOA reference architecture is also a useful tool in the design of vendor-neutral SOA solutions.

Figure 4 magnifies the services layer shown in Figure 3, and it shows that the services layer can be further divided into sublayers. It includes the services that will be delivered by a given architecture, including both composite and atomic services. Figure 3 shows how to build an SOA solution using the underlying middleware and infrastructure services provided and categorized in Figure 4.

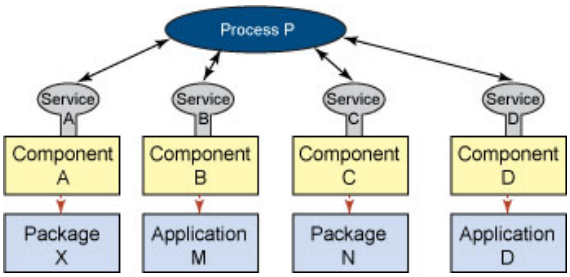
Figure 4. The middleware view of the SOA reference architecture



Layer 4. Business process layer

Compositions and choreographies of services exposed in layer 3 are defined in this layer. We use service composition to combine groups of services into flows, or we choreograph services into flows, thereby establishing applications out of services. These applications support specific use cases and business processes. To do this, visual flow composition tools can be used for design of application flows. Figure 5 shows how a business process P can be implemented using services A, B, C, and D from the services layer. Process P contains the logic for the sequence in which the services need to be invoked and executed. The services that are aggregated as a business process, or flow, can be individual services or composite services made up of individual services.

Figure 5. Services orchestration



The business process layer covers the process representation, composition methods, and building blocks for aggregating loosely coupled services as a sequencing process aligned with business goals. Data flow and control flow are used to enable interactions between services and business processes. The interaction may exist within an enterprise or across multiple enterprises.

This layer includes information exchange flow between participants (individual users and business entities), resources, and processes in a variety of forms to achieve the business goal. Most of the exchanged information may also include nonstructured and nontransactional messages. Business logic is used to form service flows as parallel tasks or sequential tasks based on business rules, policies, and other business requirements. The layer also includes information about data flows within the enterprise or across multiple enterprises.

The life-cycle management for business process orchestration and choreography is also covered in this layer. In addition to the run-time process engine (for example, WS4BPEL engine), this layer covers all aspects of composition, collaboration, compliance, process library, process service, and invocation elements.

On demand building process blocks allow a change from high-volume transactional supporting technologies to a sophisticated, much smaller footprint and less-expensive applications. In today's business solutions, business processes play a central role in bridging the gap between business and IT.

A business process captures the activities needed to accomplish a specific business goal. Typically, an enterprise uses both top-down and bottom-up approaches to assure proper business process definition. Using the top-down approach, business processes are defined by business analysts based on customers' requirements. To optimize the business process for better IT implementation, it is componentized as a reusable service that can be modeled, analyzed, and optimized based on business requirements such as quality of service (QoS) described in layer 7, flow preference, price, time of delivery, and customer preferences. Using a bottom-up approach, after creating a set of assets, we would try to leverage them in a meaningful business context to satisfy customer requirements. The flexibility and extensibility of services composition guided by business requirements and composition rules help make business process into an on demand entity for addressing different types of customer pain points by reusing services assets.

The business process layer communicates with the consumer layer (also called the presentation layer) to communicate inputs and results from the various people who use the system (end users, decision makers, system administrators) through Web portals or business-to-business (B2B) programs. Most of the control-flow messages and data-flow messages of the business process may be routed and transformed through the integration layer. The structure of the messages is most often defined by the information architecture layer. The key performance indicators (KPIs) for each task or process could be defined in the QoS and business intelligence layers. The design of service aggregations is guided by the governance layer. Of course, all the services should be represented and described by the services layer in the SOA solution stack.

From a technical perspective, dynamic and automatic business process composition poses critical challenges to researchers and practitioners in the field of Web services [3]. Business processes are driven by business requirements, which typically tend to be informal, subjective, and difficult to quantify. Therefore, it is critical to properly formulate the descriptive and subjective requirements into quantifiable, objective, and machine-readable formats in order to enable automatic business process composition. In addition, the current Web services specifications generally lack the facility to define comprehensive relationships among business entities, business services, and operations. These relationships may be important to optimize business process composition. Clearly specifying search requirements to discover the most appropriate Web services candidates remains a challenge. Last, a typical business process generally requires multiple Web services to collaborate in order to serve business requirements. Therefore, each service not only needs to satisfy individual requirements, but must also coexist with other services to fit within the

overall composed business process. This suggests that the entire business process needs to be optimized prior to execution.

Clearly, the business process layer in the SOA solution stack plays a central coordinating role in connecting business-level requirements and IT-level solution components through collaboration with the integration layer, QoS, and business intelligence layer, as well as the information architecture layer and services layer. Addressing the challenging issues that come up in the business process layer can further differentiate the proposed SOA solution stack from conceptual reference architectures proposed by other vendors.

Layer 5. Consumer layer

The consumer layer, or the presentation layer, provides the capabilities required to deliver IT functions and data to end users to meet specific usage preferences. This layer can also provide an interface for application to application communication. The consumer layer of the SOA solution stack provides the capability to quickly create the front end of business processes and composite applications to respond to changes in user needs through channels, portals, rich clients, and other mechanisms. It enables channel-independent access to those business processes supported by various application and platforms. It is important to note that SOA decouples the user interface from the components. Some recent standards such as Web Services for Remote Portlets (WSRP) Version 2.0 can be used to leverage Web services at the application interface or presentation level. Other suitable standards include SCA components, portlets, and Web Services for Remote Portlets (WSRP).

Adopting proven front-end access patterns (for example, portals) and open standards (such as WSRP) can decrease development and deployment cycle times through the use of prebuilt, proven, and reusable front-end building blocks. Use of these patterns also reduces complexity and maintenance costs through use of those common building blocks. This practice promotes a single unified view of knowledge presentation as well as a single unified entry point to the supported business processes and applications. This unified entry point integrates with other foundational services, such as security (single sign-on, for example) and trust, and significantly improves the usability of the business process and application. More specifically, it allows for the plug and play of content sources (for example, portlets) with portals and other aggregating Web applications. As a result, adopting common front-end patterns standardizes the consumption of Web services in portal front ends and the way in which content providers write Web services for portals.

Scenarios that motivate WSRP-like functionality include:

1. Portal servers providing portlets as presentation-oriented Web services that can be used by aggregation engines
2. Portal servers consuming presentation-oriented Web services provided by portal or nonportal content providers and integrating them into a portal framework

The same functionality can also be obtained through nonportal environments. WSRP allows content to be hosted in the environment most suitable for its execution while still being easily accessed by content aggregators. The standard enables content producers to maintain control over the code that formats the presentation of their content. By reducing the cost for aggregators

to access their content, WSRP increases the rate at which content sources may be easily integrated into pages for end users. It should be noted that Asynchronous JavaScript and XML (Ajax), which is used to exchange XML contents over HTTP without refreshing Web browsers, can be used to enhance SOA interaction capability with Web users.

Layer 6. Integration layer

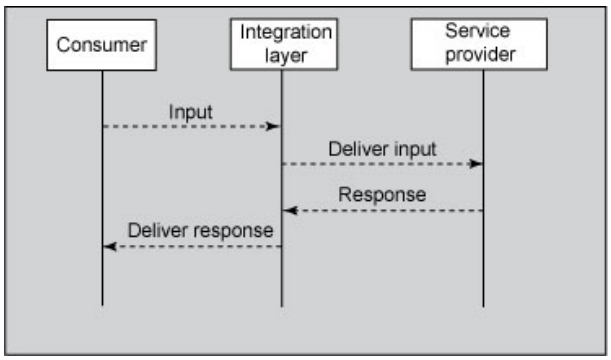
The integration layer is a key enabler for an SOA because it provides the capability to mediate, route, and transport service requests from the service requester to the correct service provider. This layer enables the integration of services through the introduction of a reliable set of capabilities. These include modest point-to-point capabilities for tightly coupled endpoint integration as well as more intelligent routing, protocol mediation, and other transformation mechanisms often provided by an enterprise service bus (ESB). Web Services Description Language (WSDL) specifies a binding, which implies the location where a service is provided. An ESB, on the other hand, provides a location-independent mechanism for integration.

The integration that occurs here is primarily the integration of layers 2 thru 4. This is the layer that provides communications, invocation, and quality of service between adjacent layers in an SOA. For example, this layer is where binding of services occurs for process execution, allowing a service to be exposed consistently across multiple customer-facing channels such as Web, IVR, Seibel client, and the like. The transformation of response to HTML (for Web), Voice XML (for IVR), XML string (for Siebel client) can be done using XSLT functionality supported through ESB transformation capability in the integration layer.

As shown in Figure 6, the integration layer does the following:

- Provides a level of indirection between the consumer of functionality and its provider. A service consumer interacts with the service provider by way of the integration layer. As a result, each service specification is only exposed through the integration layer (such as an ESB), never directly.
- Decouples consumers and providers, allowing for integration of disparate systems into new solutions.

Figure 6. Interaction diagram of the integration layer



Layer 7. Quality of service layer

Inherent in SOA are characteristics that exacerbate existing QoS concerns in computer systems. Among those characteristics are:

- Increased virtualization
- Loose coupling
- Widespread use of XML
- The composition of federated services
- Heterogeneous computing infrastructures
- Decentralized SLAs
- The need to aggregate IT QoS metrics to produce business metrics

These characteristics create complications for QoS that clearly require attention within any SOA solution.

The QoS layer provides an SOA with the capabilities required to realize nonfunctional requirements (NFRs). It must also capture, monitor, log, and signal noncompliance with those requirements relating to the relevant service qualities associated with each SOA layer. This layer serves as an observer of the other layers and can emit signals or events when a noncompliance condition is detected or, preferably, when a noncompliance condition is anticipated.

Layer 7 establishes NFR-related issues as a primary feature or concern of SOA and provides a focal point for dealing with them in any given solution. This layer provides the means of ensuring that an SOA meets its requirements with respect to reliability, availability, manageability, scalability, and security. Finally, it enhances the business value of SOA by enabling businesses to monitor the business processes contained in the SOA with respect to the business KPIs that they influence.

Layer 8. Information architecture and business intelligence layer

The information architecture and business intelligence layer ensures the inclusion of key considerations pertaining to data architecture and information architectures that can also be used as the basis for the creation of business intelligence through data marts and data warehouses. This includes metadata content, which is stored in this layer, as well as information architecture and business intelligence considerations.

> Layers 1, 6, 7, and 8 have registries associated with them. In layer 1, for example, the registry tracks the existing services that can be found in the operational layer. SOA will be built incrementally, project by project, in most cases. After one project is complete, another project can leverage the existing assets provided by the previous project. Among these assets are services (for example, Web services) created in a prior project that are available for recomposition or realization of new (composite) services. Other registries, repositories, or databases exist in other layers. For instance, the information architecture layer (layer 8) includes databases to store metadata and other data structures. The QoS layer (layer 7) will store access control lists (ACLs) or Lightweight Directory Access Protocol (LDAP) entries in registries for its operational and nonfunctional concerns.

Especially applicable to industry-specific SOA solutions, this layer captures cross-industry and industry-specific data structures, XML-based metadata architectures (that is, XML schema), and business protocols of exchanging business data. Some discovery, data mining, and analytic modeling of data is also covered in this layer.

Layer 9. Governance layer

The governance layer covers all aspects of business operational life-cycle management in SOA. It provides guidance and policies for making decisions about an SOA and managing all aspects of an SOA solution, including capacity, performance, security, and monitoring. It enables SOA governance services to be fully integrated by emphasizing the operational life-cycle management aspect of the SOA. This layer can be applied to all the other layers in the SOA solution stack. Since it helps enforce QoS and make appropriate application of performance metrics, it is well connected with layer 7.

This layer can speed the SOA solution planning and design process. The governance layer provides an extensible and flexible SOA governance framework that includes solution-level service-level agreements based on QoS and KPIs, a set of capacity planning and performance management policies to design and tune SOA solutions, and solution-level security enablement guidelines from a federated applications perspective. The architectural decisions in this layer are encapsulated in consulting practices, frameworks, architectural artifacts, documentation of SOA capacity planning, any SOA-solution SLAs, SOA performance-monitoring policies, and SOA solution-level security-enablement guidelines.

Note that we do not have a separate layer for business rules and policies. Business rules cut across all layers. For example, business process and governance layers intersect in defining the rules and policies for the business process. Consumer layer validation rules, and input and output transformations from and to that layer, must abide by some rules. These lie at the intersection point between the consumer and governance and policy layer.

Related work and usages of SOA solution stack

You can follow various paths when you use SOA solution stack to create SOA solutions. Your path to an SOA solution can be business-process-driven, tool-based architecture-driven, or data-access-based (starting with information or data services). It can start with message-based application integration through service-oriented integration, with legacy encapsulation and wrapping, or with legacy componentization and transformation.

As we apply an SOA method, such as IBM's SOMA method, every element of SOA that is identified is mapped back to the SOA solution stack providing a "dashboard view" of the SOA in progress. This view becomes useful as a communication means for business and IT stakeholders in the outcome of the SOA. In addition, the SOA solution stack can be used in the technical feasibility exploration activity of the SOMA Realization phase; in this phase, you develop extensible prototypes that test the premises of the architecture and its decisions in a risk-driven fashion.

It is important to recognize that SOA systems and solutions are designed and implemented by leveraging existing technologies and components. These existing technologies and components have an associated set of best practices that are not specifically related to SOA. J2EE applications and components can be important parts of SOA solutions, but the best practices associated with building them are not related specifically to SOA. In this article, we primarily focus on the areas that are critical success factors in building Service-Oriented Architectures.

The usage of the SOA solution stack is a key enabler for the achievement of the value propositions of an SOA. Informally, the SOA solution stack answers an architect's questions such as, "What are the layers I need to look at, building blocks I need to consider, and architectural decisions I need to make when choosing to use a set of building blocks within a given layer? Who will be using these principles and guidelines?"

The SOA solution stack applies to various types of architect-practitioners, such as enterprise architects, solution architects, and so forth. The SOA solution stack reference architecture is an abstract, logical design of an SOA, which means it answers the question "What is an SOA?" Architects can use it as a check list of layers, architectural building blocks and their relations in each layer, the options available to them, and decisions that need to be made at each layer. The layers provide a starting point for the separation of concerns needed to build an SOA.

A recurring theme for SOA projects is the applicability of the architectural style within an expanding range; SOA can start within a single project, expand to meet the needs of a line of business or a few lines of business sharing services, and then be expanded to an enterprise scale, a supply chain, or even a larger SOA ecosystem. In each case the principles of SOA tend to be applied in a similar manner. This self-similarity of the application of SOA concepts recursively within larger or smaller scopes is termed "fractal" usage of the SOA paradigm.

When we apply SOA, as defined in the above reference architecture, to a given level of an SOA ecosystem (a single project, a line of business, etc.), we typically find the need to create the same layers for each level. Thus, enterprise architecture might take the reference architecture as a blueprint that will be customized or instantiated for each line of business or each product line (depending on how the organization is structured). To participate in an SOA or services ecosystem, a company needs to have a standard reference architecture such as that depicted by the SOA solution stack in order to facilitate the integration and collaboration of architectures across companies. This way, standardization can benefit companies at the architectural level just as it benefited them at the level of data interchange using XML and XML Schema.

Summary

In this article you explored an SOA reference architecture, which provides roadmaps and guidelines for architectural, design, and implementation decisions. Additionally, it provides patterns and insights for integrating these aspects. As part of that reference architecture, reusable assets are being created to enable end-to-end, SOA-based business solutions that cover enterprise modeling, business process modeling, service modeling, as well as integration and management of business applications.

The long-term goal of the SOA solution stack is to provide templates and guidelines to help architects facilitate and automate the process of modeling and documenting the architectural layers, building blocks, options, product mappings, and architectural and design decisions that contribute to the creation of an SOA.

Acknowledgements

The authors gratefully acknowledge the input and advice received from the following colleagues: Raj Cherchatil, Arnauld Desprets, Jorge Diaz, Marc Fiammante, Donald Ferguson, Greg Flurry, Rolando Franco, Biffle French, George Galambos, Andrew Hately, Rob High Jr., Kerrie Holley, Joe Hardman, Petra Kopp, Rao Kormili, David Janson, Min Luo, Stefan Pappe, Emily Plachy, Siddarth Purohit, Robert Rafuse, Rachel Reinitz, Rick Robinson, Tony Storey, Raghu Varadan, Dan Wolfson, Bobby Wolf, Jamshid Vayghan, Olaf Zimmerman, and other members of the IBM worldwide SOA community of practice.

References

1. "Service-oriented modeling and architecture: How to identify, specify and realize your services," by Ali Arsanjani (developerWorks, Nov 2004)
2. *Note:* The infrastructure required of a SOA at run time must provide communication, invocation, and quality of service between adjacent layers in an SOA.
3. "Requirements-Driven Dynamic Services Composition for Web Services and Grid Solutions," by Liang-Jie Zhang and Bing Li. (*Journal of Grid Computing*; 2(2): 121-140. 2004)
4. "SOA programming model," by Donald Ferguson and Marcia Stockton (developerWorks, Jun 2005)
5. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language is available at http://www.w3.org/TR/2006/CR-wsdl20-20060327, by R. Chinnici, J-J. Moreau, A. Ryman, S. Weerawarana (World Wide Web Consortium, Mar 2006). The latest version (Mar 2007) of WSDL Version 2.0 Part 1: Core Language is available at http://www.w3.org/TR/wsdl20.
6. "The modular structure of complex systems," by D. L. Parnas, P. C. Clements, and D. M. Weiss. (*IEEE Transactions on Software Engineering*, SE-11(3): 259-266. 1985.)
7. *Software Architecture: Perspectives on an Emerging Discipline*, by M. Shaw and D. Garlan. (Prentice-Hall, 1996)
8. "Explicit Representation of Service Semantics: Towards Automated Composition Through a Dynamically Reconfigurable Architectural Style for On Demand Computing," by Ali Arsanjani. (proceedings of the International Conference on Web Services; pp 34 - 37; 2003.) See ICWS 2003.)

Resources

Learn

- In the Architecture area on developerWorks, get the resources you need to advance your skills in the architecture arena.
- Stay current with developerWorks technical events and webcasts.

Get products and technologies

- Download IBM product evaluation versions and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Participate in the discussion forum for this content.
- Check out Ali Arsanjani's blog, "BPM & Service-oriented Architecture: Insights and Best Practices" and get involved in the SOA community.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the authors

Ali Arsanjani, Ph.D.

Dr. Ali Arsanjani is chief architect of the SOA and Web services Center of Excellence in IBM Business Consulting Services. He is also the IBM Worldwide SOA Community leader.

Liang-Jie Zhang

Dr. Liang-Jie Zhang is a research staff member of SOA Services Research at IBM T.J. Watson Research Center, New York.

Michael Ellis

Michael Ellis is a solution architect at IBM Software Services for WebSphere, Canada.

Abdul Allam

Abdul Allam is a Certified IT Architect at IBM Business Consulting Services.

Kishore Channabasavaiah

Kishore Channabasavaiah is a lead architect in the SOA and web services Center of Excellence in IBM Business Consulting Services.