Draft Technical Standard

SOA Reference Architecture

THE Open GROUP
Making standards work®

Contents

Preface

The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow™ will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including UNIX® certification.

Further information on The Open Group is available at www.opengroup.org.

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

More information is available at www.opengroup.org/certification.

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/bookstore.

This Document

This document is the Draft Technical Standard for the SOA Reference Architecture. It has been developed by the SOA Reference Architecture project of The Open Group’s SOA Work Group. It is a draft that is made available for comment, and is not a formal Open Group publication. An Open Group Publication may eventually result, following comment, revision, and review. That publication will not necessarily reflect the contents of this draft in any way.

Trademarks

Referenced Documents

[1] Ali Arsanjani, Service-oriented modeling and architecture:How to Identify, Specify and Realize your Services, IBM developerWorks, (http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/).

[2] Ali Arsanjani, Liang-Jie Zhang, Abdul Allam, Michael Ellis, et al., “Design an SOA solution using a reference architecture”, IBM developerWorks, http://www.ibm.com/developerworks/library/ar-archtemp/

[3] Ali Arsanjani, Liang-Jie Zhang, Abdul Allam, Michael ellis, et al., “S3: A Service-Oriented Reference Architecture”, IT Professional, Volume 9, Issue 3, May-June 2007 Page(s):10 - 17”

[4] Liang-Jie Zhang, Bing Li, Requirements Driven Dynamic Services Composition for Web Services and Grid Solutions. Journal of Grid Computing 2(2): 121-140 (2004).

[5] Donald Ferguson, Marcia Stockton, SOA Programming Model, http://www-128.ibm.com/developerworks/webservices/library/ws-soa-progmodel/

[6] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, R. Chinnici, J-J. Moreau, A. Ryman, S. Weerawarana, Editors. World Wide Web Consortium, 27 March 2006. This version of the "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" Specification is available is available at http://www.w3.org/TR/2006/CR-wsdl20-20060327. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" is available at http://www.w3.org/TR/wsdl20.

[7] D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems. IEEE Transactions on Software Engineering, SE-11(3), 1985, pp. 259-266.

[8] M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, 1996.

[9] Ali Arsanjani: Explicit Representation of Service Semantics: Towards Automated Composition Through a Dynamically Re-Configurable Architectural Style for On Demand Computing. ICWS 2003: 34-37

[10] Gordon Bell, Allen Newell, Daniel Sieworek, Computer Structures: Principles and Examples, McGrawHill, 1982, Chp 3

[11] Enterprise Continuum, The Open Group Architecture Framework Version 9, , https://www.opengroup.org/architecture/togaf9-doc/arch/toc-pt5.html, TOGAF Online > Part V: Enterprise Continuum and Tools > Enterprise Continuum, 2009

[12] Definition of SOA 1.1, The Open Group SOA Work Group, https://www.opengroup.org/projects/soa/doc.tpl?gdid=10632, June 8, 2006

[13] Web Services Policy 1.5 W3C Recommendation, A. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P.Yendluri, T. Boubez, U. Yalcinalp Editors, World Wide Web

Consortium, 04 September 2007. http://www.w3.org/TR/2007/REC-ws-policy-20070904/, 2007

[14] Architecture Building Block Definition, The Open Group Architecture Framework Version 9, http://www.opengroup.org/architecture/togaf9-doc/arch/chap03.html, Definitions, 2009

[15] OASIS Open Composite Services Architecture (Open CSA) Member section, OASIS, http://www.oasis-opencsa.org/

[16] Web Services for Remote Portlets (WSRP) V1.0, OASIS, August 2003, http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf

[17] Web Services Business Process Execution Language v2.0 (WS4BPEL), OASIS Standard, April 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[18] Voice Extensible Markup Language (VoiceXML), W3C Recommendation, March 2004, http://www.w3.org/TR/voicexml20/

[19] Java 2 Platform Enterprise Edition (J2EE), Sun Microsystems, San Jose, CA, http://java.sun.com/javaee/

[20] Microsoft .net Framework, Microsoft, Redmond, WA, http://www.microsoft.com/NET

[21] XML Transformations (XSLT) version 1.0, W3C Recommendation November 1999, http://www.w3.org/TR/xslt

[22] Garrett, JJ, (2005-02-18). « Ajax : a New Approach to Web Applications ». Adaptive Path.com. . Retrieved on 2008-06-19

1 Introduction

1.1 Objective

An Service-oriented Architecture (SOA) facilitates the creation of flexible, reusable assets for enabling end-to-end SOA-based business solutions.

The usage of the SOA Reference Architecture is a key enabler for the achievement of the value propositions of a Service-oriented Architecture.

This specification presents a SOA Reference Architecture (SOA RA), which provides guidelines for making architectural, design, and implementation decisions. The goal of the SOA Reference Architecture is to provide a blueprint for creating or evaluating an architecture Additionally, it provides patterns and insights for integrating these fundamental elements of an SOA as exemplified in the layers of an SOA.

Informally, the SOA Reference Architecture is designed to answer some of the key questions and issues encountered by architects such as:

- "What are the aspects of an SOA as expressed in terms of layers that I need to consider in designing solutions based on service-oriented principles?
- What are the building blocks I need to include in each layer of my solution?,
- What are some of the key architectural decisions I need to make when designing a solution that is based on a service-oriented architecture? "
- "Which roles in a project would benefit from using these principles and guidelines?"

The SOA RA is used as a blueprint and includes templates and guidelines for architects. These facilitate and ultimately enable automation and streamlining the process of modelling and documenting the architectural layers, the architectural building blocks (ABB) within them, options for layers and ABBs, mapping of products to the ABBs and architectural and design decisions that contribute to the creation of a SOA. It is intended to support organizations adopting SOA, product vendors building SOA infrastructure components, integrators engaged in the building of SOA solutions and standards bodies engaged in the specifications for SOA.

1.2 Overview

In this specification we present the results of abstracting and using a SOA Reference Architecture based on multiple projects in different industries, commencing from 2002.

During these projects, a number of key underlying themes have come up that have been formalized into a meta-model for the SOA Reference Architecture.

This meta-model defines a number of layers, architectural building blocks, architectural and design decisions, patterns, options and the separation of concerns needed to design or evaluate an architecture.

Furthermore, these elements represent the results of applying an SOA Method to identify, specify, realize and implement services, components and flows of an end-to-end solution in the context of a service-oriented approach. The SOA Reference Architecture provides a common systems architecture [9], and the mechanism to translate it to an industry architecture and an individual solution architecture, as well as providing traceability between these specializations of architecture.

Thus the SOA RA acts as a communication vehicle for organizations to use to provide a high level specification of what an SOA’s components are and how to pick specific solutions.

The SOA Reference Architecture is based on establishing the building blocks of SOA: services, components and flows (processes) that collectively support business processes and goals. The meta-data underlying each layer and relationship between layers can further facilitate in bridging the gap between business and IT from solution modeling to solution realization.

Another major capability afforded by the SOA RA is the increase of reusability when designing and developing solution assets for rapid development, deployment and management of SOA solutions within industry or cross industries.

This document is organized as follows.

- Chapter 1 provides a general introduction.
- Chapter 2 provides an overview of the principles on which the SOA RA was derived
- Chapter 3 provides an introduction to the SOA Reference Architecture
- Chapter 4 over views the Reference Architecture
- Chapter 5 reviews the layers of the Reference Architecture in detail
- Chapter 6 illustrates applications of the Reference Architecture

1.3 Conformance

Intentionally left blank in this version

1.4 Terminology

Can	Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to this document. An application can rely on the existence of the feature or behavior.
Implementation-dependent	(Same meaning as "implementation-defined".) Describes a value or behavior that is not defined by this document but is selected by an implementer. The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations. The implementer shall document such a value or behavior so that it can be used correctly by an application.

Legacy	Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable applications. New applications should use alternative means of obtaining equivalent functionality.
May	Describes a feature or behavior that is optional for an implementation that conforms to this document. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations. To avoid ambiguity, the opposite of "may" is expressed as "need not", instead of "may not".
Must	Describes a feature or behavior that is mandatory for an application or user. An implementation that conforms to this document shall support this feature or behavior.
Shall	Describes a feature or behavior that is mandatory for an implementation that conforms to this document. An application can rely on the existence of the feature or behavior.
Should	For an implementation that conforms to this document, describes a feature or behavior that is recommended but not mandatory. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations. For an application, describes a feature or behavior that is recommended programming practice for optimum portability.
Undefined	Describes the nature of a value or behavior not defined by this document that results from use of an invalid program construct or invalid data input. The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.
Unspecified	Describes the nature of a value or behavior not specified by this document that results from use of a valid program construct or valid data input. The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.
Will	Same meaning as "shall"; "shall" is the preferred term.

1.5 Future Directions

Subsequent drafts of the SOA Reference Architecture will identify the architectural building blocks and associated capabilities in each of the layers. It will also include examples of application of the reference architecture. There is no commitment to any particular additional content and other content not mentioned here may be added.

2 Motivation

Over the past 25 years Software Architecture has grown rapidly as a discipline. With the pioneering work of Bell, Newell and Sieworek [8] (PMS notation and the birth of ADLs), Shaw and Garlan on Software Architecture [6], and later additions, we recognize the need for architectural patterns and styles. An architectural style is the combination of distinctive features in which architecture is performed or expressed [10]. This can be visualized as a set of components (we will call architectural building blocks to distinguish the generic nature of these), connectors, constraints [6], composition, containers and configurations. Arsanjani refers to these as the 6C's of an architectural style [7]. We know that a software system often has a predominant architectural style for its design. It has been shown that modularity or decoupling facilitates the creation of complex systems [5] such as that required by today's business applications.

In recent years, the decoupling of interface from implementation at the programming level has been elevated to an architectural level by loosely coupling the interface of a service consumed by a Service Consumer from its implementation by a Service Provider and by decoupling the implementation from its binding. This style of architecture has come to be known as service-oriented architecture (SOA), where 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.

Thus, a SOA enables business and IT convergence through agreement on a (contract consisting of 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 also provides the mechanisms to externalize variations [7] of quality of service in declarative specifications such as WS-Policy [11] and related standards.

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

- *Reducing Cost:* Through providing the opportunity to consolidate redundant application functionality and decouple functionality from obsolete and increasingly costly applications while leveraging existing investments.
- *Agility:* Structure business solutions based on a set of business and IT services in such as way as to facilitate the rapid restructuring and reconfiguration of the business processes and solutions that consume them.
- *Increasing Competitive Advantage:* Provide 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. Potentially increase market share and business value by offering new and better business services.
- *Time to Market:* Deliver business-aligned solutions faster by allowing the business to decide on the key drivers of a solutions and allowing IT to rapidly support and implement that direction.
- *Consolidation:* Integrate across silo-ed solutions and organizations, reduce the physical number of systems and enable consolidation of platforms under a program of

"graceful transition" from legacy spaghetti dependencies to a more organized and integrated set of coexisting systems.

However, significant challenges in creating a SOA solution still remain. Correct design, solution element selection and combination, modelling of services, governance, interoperability, and the ability to identify different components are key to the effective design, usage and evolution of SOA. For example, from a technical perspective, the architect needs to answer questions such as:

- "What are the considerations and criteria for producing a SOA solution?"
- " How can a SOA solution be organized as an architectural framework with inter-connected architectures and transformation capabilities?
- How can a SOA solution be designed in a manner that maximizes asset reuse?
- How can automated tools take the guess work out of architecture validation and capacity planning?"

In order to address these issues, this specification presents a reference architecture for SOA-based solutions. It provides a high level abstraction of a SOA partitioned and factored into layers, each of which addresses a specific subset of characteristics and responsibilities that relate to unique value propositions within a SOA. As stated above, underlying this layered architecture is a meta-model consisting of layers, architectural building blocks (ABB), relations between ABB's and between layers, interactions patterns, options and architectural decisions. These will guide the architect in the creation and evaluation of the architecture. Note that an architectural building block represents a basic element of reusable functionality that can be realized by one or more components or products; examples of the responsibilties of an ABB include: service definition, mediation, routing, etc.

3 Key Principles

The reference architecture has been defined and refined with consideration for the following principles:

1. The SOA RA should be a generic solution that is vendor neutral and
2. The SOA RA is based on a model of standards compliance.
3. The SOA RA should be capable of being instantiated to produce
    a. intermediary industry architectures and
    b. concrete solution architectures.
4. It should address multiple stakeholder perspectives.
    a. For Organizations implementing the RA within their enterprise:
        i. The reference architecture should be generic enough to be independent of vendor solutions.
        ii. The RA should define standard building blocks, architectural decisions and other attributes to create a framework of understanding sufficient to enable an assessment of conformance.
    b. For Product Vendors:
        i. The reference architecture should provide a set of standards and enough specificity that they can use it to drive evaluation of compliance with those underlying standards.
    c. For Integrators:
        i. The integrator should be able to use it as a model to define specific constraints and directions for SOA implementations.
    d. For Standards Bodies:
        i. The RA should provide a reference against which they can extend standards, or provide guidelines, and more detailed levels of specificity etc.
5. The manner in which it is instantiated should be determined by the user of the RA. It should use the fewest number of layers to depict the possible combinations and elements of a service-oriented architecture.

4 Overview of the Layers of the SOA Reference Architecture

The SOA Reference Architecture has nine layers which represent nine of the key clusters of considerations and responsibilties that arise in the process of designing and SOA solution. Also, each layer is designed to materialize and reinforce each of the various perspectives of SOA business value.

For each layer, it is practical to postulate two aspects: logical and physical. The logical aspect includes all the architectural building blocks, design decisions, options, KPIs, etc; while the physical aspect of each layer includes the realization of each logical aspect using technology, standards and products, that are determined by taking into consideration the different architectural decisions that are necessary to be made to realize and construct the architecture .

This specification provides specific focus on the logical aspects of the SOA Reference Architecture, and provides a model for including key architectural considerations and making architectural decisions through the elements of the meta-model.

Three of the layers address the implementation and interface with a service (the Operational Systems Layer, the Service Components Layer and the Services Layer). Two of them support the consumption of services (the Business Process Layer and the Consumer Layer). Four of them support cross-cutting concerns of a more "non-functional" nature (the Information Architecture, Quality of Service, Integration and Governance Layers). The SOA RA as a whole provides the framework for the support of all the elements of a SOA, including all the components that support services, and their interactions.

This logical view of the SOA Reference Architecture addresses the question, "If I build a SOA, what would it look like and what abstractions should be present?"

The SOA Reference Architecture enumerates the fundamental elements of a SOA solution and provides the architectural foundation for the solution.

As shown in Figure 1, the meta-model of the SOA Reference Architecture includes the following elements:

- *Layer:* an abstraction which contains a set of components such as ABBs, architectural decisions, interactions among components and interactions among layers.
- *ABB (Architectural Building Block):* a constituent of the architecture model that describes a single aspect of the overall model [12]. Each layer can be thought to contain a set of ABBs that define the key responsibilties of that layer. In addition, ABBs are connected to one another across layers and thus provide a natural definition of the association between layers. In this reference architecture each ABB resides in a layer, supports capabilities, and has responsibilities. It contains attributes, dependencies, constraints and relationships with other ABBs in the same layer or different layer.
- *Method activity:* a set of steps that involve ABBs within a layer. The method activity provides a dynamic view of how different ABBs within a layer will interact. Method

activities can also be used to describe the interaction between layers, so that an entire interaction from a service invocation to service consumption is addressed.

- *Options:* a collection of possible choices available in each layer that impact other artifacts of a layer. Options are the basis for architectural decisions within and between layers, and have concrete standards, protocols, and potentially instantiable solutions associated with them. An example would be that SOA Services can be SOAP or REST style are viable options. Which option is selected leads to an architectural decision.
- *Architectural decision:* a decision derived from the options. The architectural decision is driven by architectural requirements, and involves governance rules and standards, ABBs, KPIs (Key Performance Indicators) and NFRs (Non Functional Requirements) to decide on standards and protocols to define a particular instance of an Architectural Building Block. This can be extended, based on the instantiation of the Reference Architecture to the configuration and usage of ABBs. Existing architectural decisions can also be reused by other layers or ABBs
- *Interaction pattern:* an abstraction of the various relationships between ABBs, This includes diagrams, patterns, pattern languages and interaction protocols.
- *KPI:* key performance indicator. A key performance indicator may act as input to an architectural decision.
- *NFR:* non-functional requirement. An NFR may act as input to an architectural decision. NFRs help address Service Level Agrrement attributes, (e.g. response time, etc.) and architectural cross-cutting concerns such as security.
- *Enabling Technology:* a technical realization of ABBs in a specific layer.
- *Information Model:* a structural model of the information associated with ABBs including data exchange between layers and external services. The information model includes the meta-data about the data being exchanged.

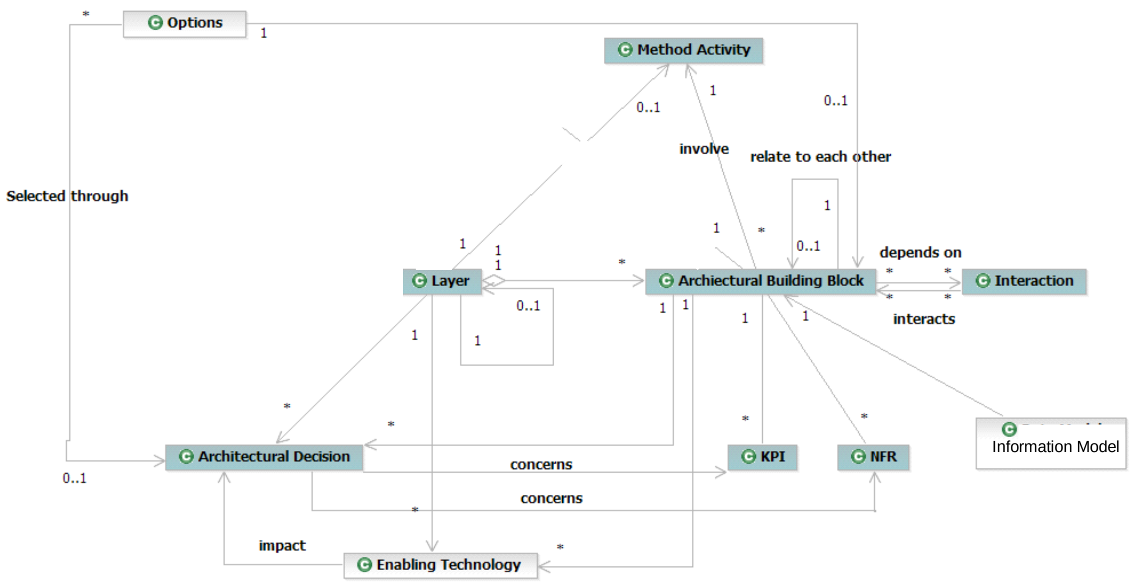

Figure 1: A Meta-model for instantiating the SOA Solution Reference Architecture for a given solution

The following architectural diagram (Figure 2) depicts a SOA as a set of logical layers. Note that one layer does not solely depend upon the layer below it and is thus named a partially-layered architecture: e.g., a consumer can access the business process layer as a service or the service layer directly, but not beyond the constraints of an SOA architectural style. For example, a given SOA solution may exclude a Business Process layer and have the Consumer Layer interacting directly with the Service Layer. Such a solution would not benefit from the business value proposition associated with the Business Process layer however that value could be achieved at a later stage by adding the layer. In this sense the SOA Reference Architecture represents SOA with a partially layered architecture. The degree to which a given organization realizes the full SOA Reference Architecture will differ according to the level of SOA maturity they exhibit.

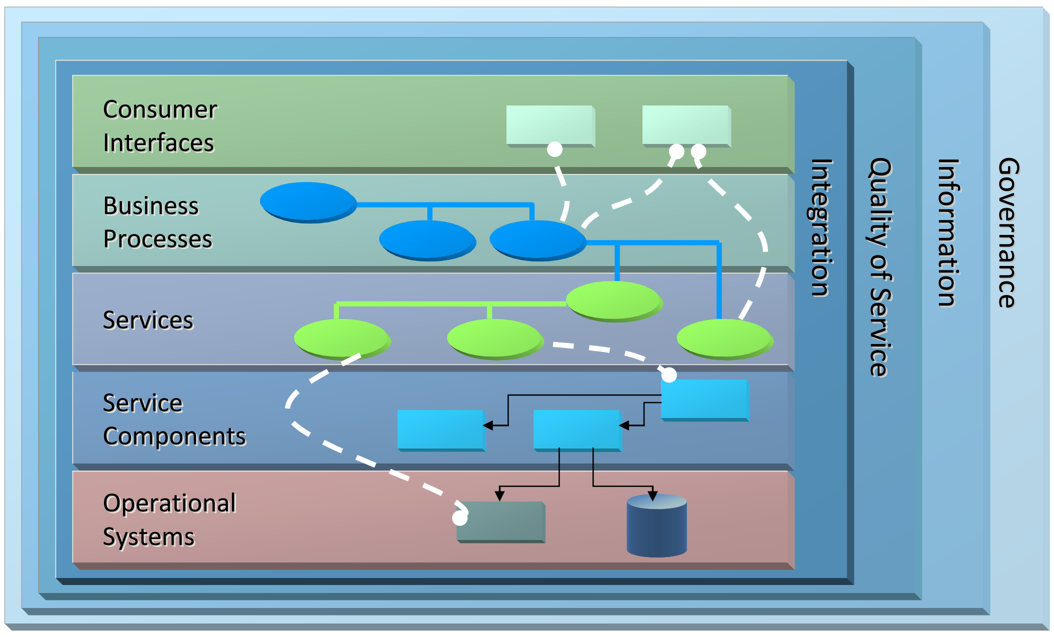

Figure 2: Layers of the SOA Reference Architecture: Solution View

Figure 2 illustrates the multiple separations of concern in the 9 layers of this reference architecture. The SOA Reference Architecture does not assume that the provider and the consumer are in one organization, and supports both SOA within the enterprise as well as across multiple enterprises. The need for both intra-and inter-enterprise SOA is important, with the role of SOA as the foundation of cloud computing and SaaS.

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 which may have different lines of business use this architectural style (where one is the consumer and the other is the provider), customizing it for their own needs and integrating and interacting among themselves; in such a case there is still real value in maintaining a decoupled consumer/provider relationship. The lower layers (Services, Service Components and Existing Application Assets) are concerns for the provider and the upper ones (Services, Business Processes and Consumers) are concerns for the consumer. Below we will describe each layer and in the subsequent sections, describe the relationships between the layers.

Note that there are five horizontal layers that are more functional in nature and relate to the functionality of the SOA solution. The vertical layers are non-functional in nature and support various cross-cutting concerns of the SOA architectural style.

5 Description of Layers

5.1 Assumptions

An SOA is defined by the set of functional and non-functional requirements that constrain it. Functional requirements are business capabilities imperative for business operations including business processes, the business and IT services, the components and underlying systems that implement those services. Non-functional service aspects include: security, availability, reliability, manageability, scalability, latency, governance and integration capabilities etc.

The underlying requirements which determine the capabilities that the SOA supports are determined by:

1. A set of service requirements which includes business (aka functional) and non functional requirements on a service
2. Service requirements result in the documented capability that a service needs to deliver or is expected to deliver.
3. The provider view of a service requirement is the business and technical capability that a given service needs to deliver given the context of all of its consumers.
4. 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 fulfilment of any service requirement may be achieved through the capabilities of a combination (one or more) of layers in the SOA Reference Architecture.

Services themselves have a contract element and functional element. The contract defines what the service does for consumers, while the functional element implements what a service contracts to provide. The service contract is integrated with the underlying functional element through a component which provides a binding. This model addresses services exposing capabilities implemented through legacy assets, new assets, services composed from other services or infrastructure services.

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 SOA Reference Architecture is a key aspect in developing a service-oriented architecture for an enterprise [2][3].

5.2 Layer 1: Operational Systems Layer

This layer captures the new and existing organization infrastructure, including those involving actors, needed to support the SOA solution. This includes:

1. All infrastructures to run the SOA and its components.

2. All operational and runtime hosting of underlying systems components, both physical and infrastructural.
3. All assets required to support the functionality of the service in the SOA, including custom or packaged application assets, new services, services created through composition or orchestration, infrastructure services, etc...

Thus the capabilities supported by the Operational Systems layer including integration of infrastructure services into the SOA, the reuse of existing assets composed within it, and the infrastructural elements required for running the SOA. Thus from a SOA perspective, the Operational Systems Layer, enables organizations to integrate in a perimeter-less, cross-organizational manner (as in the case of SaaS applications), the integration of infrastructure services used in the Cloud, and the leveraging of existing assets in custom and packaged applications (forming the base for service reuse).

This directly influences the overall cost of implementing SOA solutions within enterprises.

It is important to note that a service executes its functionality through assets in this layer. For example a patient record update service that contracts to update patient records, does so using different components running in application assets hosted in the operational systems layer.

A number of existing software systems are part of this layer. Those systems include:

- Existing monolithic custom applications including J2EE [19]and .Net [20] applications
- Other services
- Legacy applications and systems
- Existing transaction processing systems
- Existing databases
- Existing package applications and solutions including ERP and CRM packages

5.3 Layer 2: The Service Component Layer

This layer contains software components, each of which provide the implementation or “realization” for a service, or operation on a service; hence the name Service Component. Service components reflect the definition of the service they represent, both in its functionality and its quality of service. They “bind” the service contract to the implementation of the service in the operational systems layer. Service components are hosted in containers which support a service specification.

The service component layer manifests the IT conformance with each service contract 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 (ensures quality of service and service level agreements)

- Enables business flexibility by supporting the functional implementation of IT flexible services, their composition and layering
- IT flexibility by strengthening the decoupling in the system. Decoupling is achieved by hiding volatile implementation details from consumers.

Refer to the example in Figure 3, where ”Service A” is implemented using a combination of behaviour from 3rd 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 and it is the providers’ responsibility to ensure that such compliance is achieved. The details of the realization, however, are of no consequence to Application B. Service Component A acts as a service implementation façade, it aggregates available system behaviour and gives the provider an enforcement point for service compliance. Application B invokes and interoperates with a service contract and specification defined in the interface in Service A.

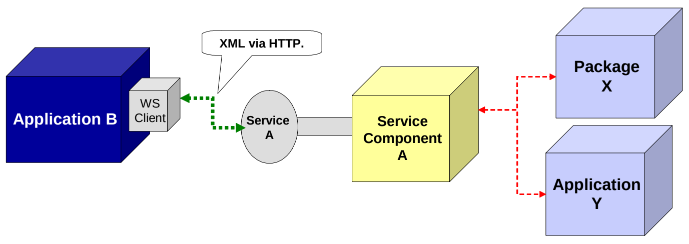

Figure 3: Service Component as a Facade

Subsequently, the Provider organization may decide to replace “Package X” with “Package M”. The required modifications are encapsulated in Service Component A 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.

5.4 Layer 3: Services Layer

This layer consists of all the services defined within the SOA. The service layer can be thought of as containing the service descriptions (design time and business architectural assets, as well as runtime service contract descriptions) and the container for implementing the services.

The specification provides consumers with sufficient detail to invoke the business functions exposed by a provider of the service; ideally this may be 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 WSDL description [4]. This information is not necessarily WSDL.

The service specification may 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 in the set implying that a significant successor/predecessor relationship exists between them.

Thus, 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 via well-defined interfaces of the “Services Layer”. The service layer also provides for the mechanism to take enterprise scale components, business unit specific components and in some cases 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 get exported out as service descriptions in this layer, where services exist in isolation (atomic) or as composite services.

For example, a “service container” which the services are hosted in and invoked from, is also a part of the Services Layer. The service container is compliant with the standards for service specification being supported by the service and runs on a hosting platform in the operational systems layer.

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

The layers and their underlying building blocks in the target architecture may be defined according to the service identification activities which may be defined through three complementary techniques of domain decomposition, existing asset analysis and goal-service modelling to identify, specify, and realize services, components and flows [1]. They represent, therefore, the heart of the SOA value proposition – improved agility via the decoupling of business and IT. The quality of these service definitions will have a significant impact on the benefit of a given SOA effort.

Services are accessible independent of implementation and transport. This allows a service to be exposed consistently across multiple customer facing channels such as Web, interactive voice response (IVR), etc. The transformation of response to HTML (for Web), Voice XML [18] (for IVR), and XML string (for XML client) can be done via XSLT [21] functionality supported through enterprise services 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, i.e. the 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 back to a business process and as such do not warrant the rigorous analysis required for business services.

In addition to being an important template for defining a SOA solution at a logical level, the SOA Reference Architecture is also a useful tool in the design of vendor neutral SOA solutions. This is because it enables the objective identification of SOA infrastructure requirements. The SOA Reference Architecture provides a well-factored decomposition of the SOA problem space which allows architects to focus on those parts of a SOA solution that are important in the context of the problem they are solving and to map the required capabilities onto vendor product capability – rather than try and reverse engineer a SOA solution architecture from the capability of a particular vendor’s products. This set of requirements can be used to better leverage the various capabilities provided by a mix of different vendors who may offer the same architectural building block. Using the same SOA Reference Architecture we can deliver SOA business services based on the same deployment framework.

5.4.1 Service Types and a Middleware View of the SOA Reference Architecture

The relationship between Figure 2 and Figure 4 is that Figure 4 is a further elaboration of the Services layer in Figure 2. The Services Layer includes aspects of the integration, information, and quality of service and governance layers. It includes the services that will be delivered by a given architecture; including both composite and atomic services.

Figure 2 depicts a Solution View of how you would build a SOA solution using the underlying middleware and infrastructure service provided and categorized by Figure 4.

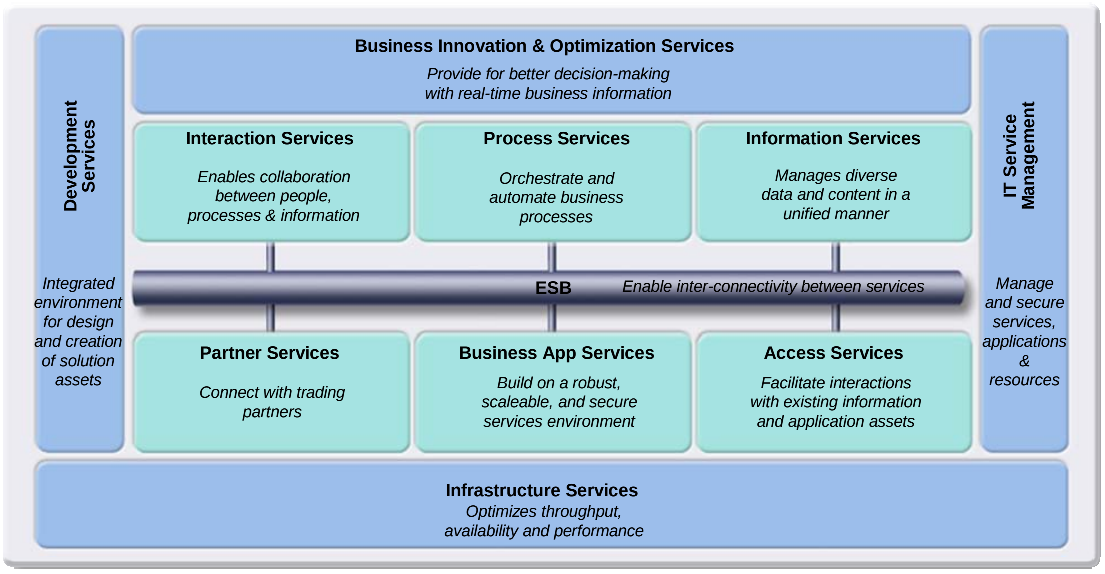
Business Innovation & Optimization Services
Provide for better decision-making with real-time business information
Development Services
Integrated environment for design and creation of solution assets
Interaction Services
Enables collaboration between people, processes & information
Process Services
Orchestrate and automate business processes
Information Services
Manages diverse data and content in a unified manner
IT Service Management
Manage and secure services, applications & resources
ESB
Enable inter-connectivity between services
Partner Services
Connect with trading partners
Business App Services
Build on a robust, scaleable, and secure services environment
Access Services
Facilitate interactions with existing information and application assets
Infrastructure Services
Optimizes throughput, availability and performance

Figure 4: The Middleware View of the SOA Reference Architecture

5.5 Layer 4: Business Process Layer

In a service oriented world, business capabilities are realized by business processes, or collections of business processes. These business processes include service orchestrations and compositions, and the ability to insert "human intervention" and support long-lived transactions.

In particular, compositions and choreographies of services exposed in layer 3 are defined in this layer. The evolution of service composition into flows, or choreographies of services bundled into a flow act together to establish an application. These applications support specific use cases and business processes. Here, visual flow composition tools can be used for design of application flow. Figure 5 shows how a Business Process "P" can be implemented using Services A, B, C and D from 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 can be an individual service or a composite service.

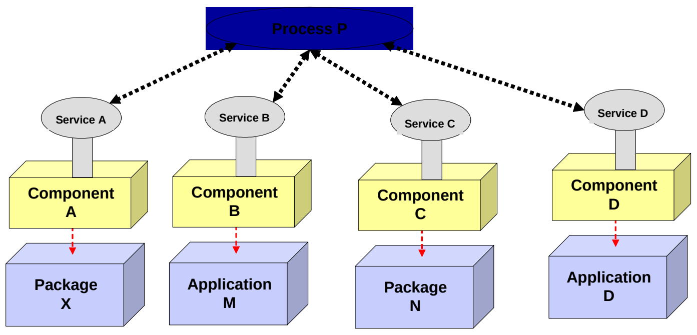

Figure 5: Services Orchestration

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 variety of forms to achieve the business goal. Most of the exchanged information may also include non-structured and non-transactional messages. The business logic is used to form service flow as parallel tasks or sequential tasks based on business rules, policies, and other business requirements.

From the data flow perspective; the business context and metadata are used to support the aggregation of services within an enterprise for business process orchestration or across multiple enterprises for business process choreography.

The lifecycle management for business process orchestration and choreography are also covered in this layer. In addition to the run-time process engine (e.g. WS4BPEL [17] engine), this layer will cover all aspects of composition, collaboration, compliance, process library, process service, and invocation elements.

Building process blocks on demand with reduced cost allow the supporting technology changes from being high volume/transactional to sophisticate but much smaller footprint applications. In fact, business process captures the activities needed to accomplish a certain business goal. In today's business solutions, business processes have played a central role in bridging the gap between business and IT.

Using the top-down approach, business processes are defined by business analysts, based on customers' requirements. In order to optimize the business process for better IT implementation, we need to componentize a business process as reusable services that can be modelled, analyzed, and optimized based on business requirements such as QoS (historical data described in layer 7), flow preference, price, time of delivery, and customer preferences. From the bottom-up approach, after we have created a set of assets, we would like 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 enable business process on demand for addressing different types of customer pain points by reusing services assets.

From the interaction perspective, the business process layer communicates with the consumer layer (a.k.a. presentation layer) to communicate inputs and results with role players (e.g. end user, decision makers, system administrator, etc.) through Web portals or business-to-business (B2B) programs. Most of the control flow messages and data flow messages of the business process maybe routed and transformed through the integration layer. The structures of the messages are most often defined by the data architecture layer. The KPIs for each task or process could be defined in QoS and business performances layer. 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 Reference Architecture.

From the technical perspective, dynamic and automatic business process composition poses critical challenges to researchers and practitioners in the field of Web services [2]. 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 facility to define comprehensive relationships among business entities, business services, and operations. These relationships may be important to optimize business process composition. How to clearly specify search requirements to discover the most appropriate Web services candidates remains a challenge. Lastly 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 also needs to coexist with other services in order to fit within the overall composed business process. This suggests that the entire business process needs to be optimized prior to execution.

In summary, the business process layer in the SOA Reference Architecture plays a central coordinating role in connecting business level requirements and IT-level solution components through collaboration with integration layer, QoS and business performance management layer, as well as data architecture layer and services layer. Addressing those challenging issues are being covered in this business process layer to further differentiate the proposed SOA Reference Architecture with other conceptual Reference Architectures from other vendors.

5.6 Layer 5: Consumer Layer

SOA supports a client-independent, channel agnostic set of functionality which is separately consumed and rendered through one or more channels. This ability empowers organizations to use SOA to improve quality, consistency and support enhanced reuse.

The consumer layer provides the capabilities required to deliver IT functions and data to end-users that meet specific usage preferences through channels, portals, rich clients (mashups and Ajax [22]) as well as being an interface for application to application communication, and to other services and channels (e.g. IVRs). Some recent standards such as Web Services for Remote Portlets version 2.0 (WSRP) [16] may indeed leverage web services at the application interface or presentation level. It is also important to note that SOA decouples the user interface from the components. Examples include Service Component Architecture (SCA) Components [15], portlets, WSRP and B2B.

The Consumer Layer of the SOA Reference Architecture provides the capability to quickly create the front end of the business processes and composite applications to respond to changes in the marketplace. It enables channel independent access to those business processes supported by various application and platforms.

By adopting proven front-end access patterns (e.g. portals) and open standards (e.g. WSRP), it can decrease development and deployment cycle times through the use of many pre-built, proven and reusable front end building blocks. It also reduces complexity and maintenance costs with those common building blocks. In this way, it promotes a single unified view of knowledge presentation, a single unified entry point to the supported business processes/applications that integrates with other foundational services such as security (single sign-on, for example) and trust, and significantly improve the usability of the business process/application.

More specifically, it allows for the plug-and-play of content sources (e.g., portlets) with portals and other aggregating web applications. It thereby standardizes the consumption of services in portal front ends and the way in which content providers write services for portals. Scenarios that motivate WSRP-like functionality include: (1) Portal servers providing portlets as presentation-oriented services that can be used by aggregation engines; (2) Portal servers consuming presentation-oriented services provided by portal or non-portal content providers and integrating them into a portal framework. The description also applies generally to non-portal 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 is noted that Asynchronous JavaScript and XML (Ajax), a way of exchanging XML contents over HTTP without refreshing Web browsers, can be used to enhance SOA interaction capability with Web users.

5.7 Layer 6: Integration Layer

The integration layer is a key enabler for a SOA as it provides the capability to mediate, transform, route and transport service requests from the service requester to the correct service provider. Thus it supports the capabilities required for SOA components to integrate into the routing/transportation/transformation ABBs, as well as the capabilities for routing, transformation and transportation. It also provides the support of a common business rules capability which is used to ensure that common business rules are applied across all the different layers in the architecture.

This layer enables the integration of services through the introduction of a reliable set of capabilities. These can start with modest point-to-point capabilities for tightly coupled endpoint integration and cover the spectrum to a set of much more intelligent routing, protocol mediation, and other transformation mechanisms often described as, but not limited to, an Enterprise Service Bus (ESB). WSDL specifies a binding, which implies location where a service is provided, and is one of the mechanisms to define a service contract. 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 (the "functional" layers). For example, this is where binding (late or otherwise) of services occurs for process execution. This allows a service to be exposed consistently across multiple customer facing channels such as Web, IVR,XML client etc. The transformation of response to HTML (for Web), Voice XML (for IVR), XML String, can be done via XSLT functionality supported through ESB transformation capability in the Integration layer.

As shown in Figure 6, the Integration layer:

- Provides a level of indirection between the consumer of functionality and its provider. A service consumer interacts with the service provider via the Integration Layer.

Hence, each service interface is only exposed via the integration layer (e.g. ESB), never directly;

- Consumers and providers are decoupled, this decoupling allows integration of disparate systems into new solutions.

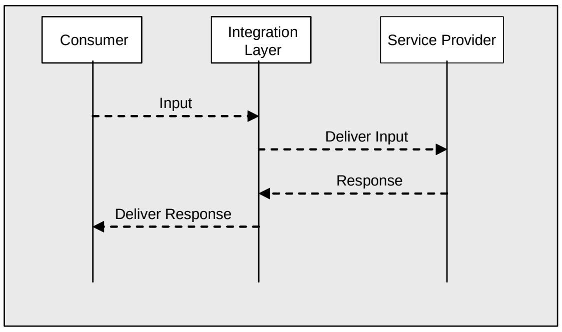

Figure 6: Interaction Diagram of Integration Layer

Business Rules and the integration layer:

As mentioned earlier, Business Rules are a cross-cutting architectural concern. They need to be consistently applied across the multiple layers in the SOA, or, over time, there is a tendency to develop rule divergence and lose the consistency that SOA brings. The integration layer provides a common business rules capability that can be used by the ESB (the component in the integration layer which mediates, routes and transforms data) and the other layers in the SOA RA.

The integration layer also incorporates the support for runtime service virtualization using a registry. Service virtualization virtualizes (decouples) the location of services for consumers of services. Services are exposed to consumers through a registry, but the exact location decoupled, to support versioning, change of service locality and administration, without impacting the consumer.

To summarize, the integration layer supports a common rules capability, the ability to integrate into the SOA, the concept of an ESB (which supports routing, mediation and transformation), and a registry capability for service virtualization.

5.8 Layer 7: Quality of Service Layer

Inherent in SOA are characteristics that exacerbate existing QoS concerns in computer systems: 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 etc. are the nature of SOA. These characteristics create complications for QoS that clearly require attention within any SOA solution.

The Quality of Service layer provides the service SOA solution lifecycle processes with the capabilities required to ensure that the defined policies and NFRs are adhered to. This layers deals with issues like:

- Monitoring and Capture of service and solution metrics in an operational sense and signalling non-compliance with non-functional requirements (NFRs) relating to the salient service qualities and policies associated with each SOA layer.

Service metrics are captured and connected with individual services to allow service consumers to evaluate service performance, creating increased service trust levels.

This layer serves as an observer of the other layers and can create events when a non-compliance condition is detected or (preferably) when a non-compliance condition is anticipated.

The QOS layer establishes NFR related issues as a primary feature/concern of SOA and provides a focal point for dealing with them in any given solution. It provides the means of ensuring that a SOA meets its requirements with respect to e.g.:

- reliability
- availability
- manageability
- scalability
- 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.

In particular, SOA security, a significant issue with SOAs' due to their potential perimeter-less nature as opposed to traditional, web-based, "within the firewall", perimeter based security is a capability realized by the QoS layer.

5.9 Layer 8: Information Architecture Layer

This layer includes information architecture, business intelligence, meta-data considerations and 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 meta-data content that is stored in this layer.

Especially, for industry-specific SOA solutions, this layer captures all the common cross-industry and industry-specific data structure, XML-based meta-data Architectures (e.g. XML schema), and business protocols of exchanging business data. Some discovery, data mining, and analytic modelling of data are also covered in this layer. These common structures may be standardized for the industry or organization.

5.10 Layer 9: Governance Layer

SOA Governance ensures that the services and SOA solutions within an organization are adhering to the defined policies, guidelines and standards that are defined as a function of the objectives, strategies and regulations applied in the organization. SOA governance activities shall conform to Corporate, IT & EA governance principles & standards. The Governance layer will be adapted to match and support the target SOA maturity level of the organization.

The governance layer includes both SOA governance (governance of processes for policy definition and enforcement) as well as Service governance (service life-cycle). This covers the entire lifecycle of the services and SOA solutions (i.e. both design and runtime) as well as the portfolio management of both the services and SOA solutions managing all aspects of services and SOA solutions (e.g. SLA, capacity and performance, security and monitoring).

This layer can be applied to all the other layers in the SOA Reference Architecture. From QoS and performance metrics perspective, it is well connected with QOS layer. From a service lifecycle and design time perspective it is connected with the Service layer. . From a SOA solution lifecycle perspective it is connected to the Business Process layer.

The goal of the SOA Governance layer is to ensure consistency of the Service and Solution portfolio and lifecycles processes. In this layer, the extensible and flexible SOA governance framework will ensure that all aspects of SOA are managed and governed such as:

- Service Level Agreements based on QoS and KPIs
- Capacity and Performance management policies
- Security enablement

The information in this layer that is collected and made available consists of e.g.

- Guidelines for SOA Governance
- Guidelines for Service and SOA Solution lifecycle and portfolio management
- Best practices
- Policies (e.g. security)
- Standards
- Service and SOA solution roadmaps
- Compliance, Dispensation and Communication documentation

6 Related Work and Usages of the SOA Reference Architecture

The SOA RA provides a mechanism for use in a variety of scenarios:

1. For organizations adopting SOA
2. For organizations building SOA components (SOA product vendors)
3. For organizations providing services in the construction of SOAs (integrators)
4. For organizations building standards centered around the specification of SOA standards.

For organizations adopting SOA, it provides a number of uses, including using the SOA Reference Architecture to create SOA solutions including

- business process driven,
- tool-based architecture-driven,
- message-based application integration through
  - service-oriented integration,
  - data access-based (information or data services),
  - legacy encapsulation and wrapping,
- and legacy componentization and transformation.

As we apply an SOA Modelling and Delivery Methods every element of SOA that is identified is mapped back to the SOA Reference Architecture providing a “dashboard view” of the SOA in progress; useful as a communication means for various business and IT stakeholders.

In addition, the SOA Reference Architecture is used to define capabilities and the technical feasibility of solutions. This usage is focused on as a realistic technique of identifying key technical extensible prototypes that test the premises of the architecture and its decisions in a risk driven fashion.

The SOA Reference Architecture provides a checklist of key elements that must be considered when you build your SOA: mandatory as well as optional layers, attributes, architectural building blocks, design decisions and interaction patterns.

It is important to recognize that SOA solutions are designed and implemented by leveraging existing techniques and technologies. They have an associated set of best practices that are not specifically related to SOA. For example, writing robust J2EE applications and components are an important part of building SOA solutions. In this specification, we just focus for the most part on the areas that are critical success factors in building service-oriented architectures.

The SOA Reference Architecture applies to various types of practitioners such as Enterprise Architects, Solution Architects, etc. The SOA Reference Architecture is an abstract, logical design of a SOA. Thus it answers the question of “What is a SOA?” Architects can use it as a checklist of layers, architectural building blocks and their relations in each layer, the options available, decisions that need to be made at each layer. The layers provide a starting point for the separation of concerns needed to build a SOA.

A recurring theme in the context of SOA projects has been the applicability of SOA within multiple areas of increasing scope: a single project, a line of business, a few lines of business sharing services, enterprise scale, supply-chain (value-net) and 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 granularity of a SOA Eco-system, we will typically find the need to create the same layers for each level of granularity. Thus, Enterprise Architecture might use the SOA Reference Architecture as an SOA solution template 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 a SOA or Services Eco-system, a company would need to have a standard reference Architecture such as that depicted by the SOA Reference Architecture, in order to facilitate the integration and collaboration of architectures across companies. Thus standardization would benefit companies at the architectural level just as it has benefited them at the level of data interchange via XML and XML Schema.

Glossary

Note to Reviewers:

> For like terms, this glossary will use the Open Group SOA Ontology and the glossaries of other Open Group SOA working groups.

Term	Definition