ISSN No. 0976-5697

Volume 4, No. 2, Jan-Feb 2013

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

Service Oriented Architecture Governance:A Review on SOA Governance Aspects and Comparitively Study of IT Govenance and SOA Governance

Soniya Goyal
Computer Science & Engineering
Poornima Group of Institutions Jaipur,India
goyal_soniya@rediffmail.com

Abstract: **T**his Paper Reviews the SOA governance and its aspects. SOA governance is a concept used for activities related to exercising control over services in a service-oriented architecture (SOA).In the first section we describe a general SOA. In section 2 we describe SOA governance, in section 3 we compare IT Governance and SOA Governance, in section 4 we describe SOA Governance aspects. Finally, in the last section we summarize.

Keywords: Service Oriented Architecture, SOA Governance, IT Governance,

I. INTRODUCTION

A. Defining Service Oriented Architecture:

Software Oriented Architecture is an emerging approach that addresses the requirements of loosely coupled, standard based and protocol independent distributed computing.[10]**.**SOA is an architectural paradigm and discipline that may be used to build infrastructures enabling those with needs (consumers) and those with capabilities (providers) to interact via services across disparate domains of technology and ownership.

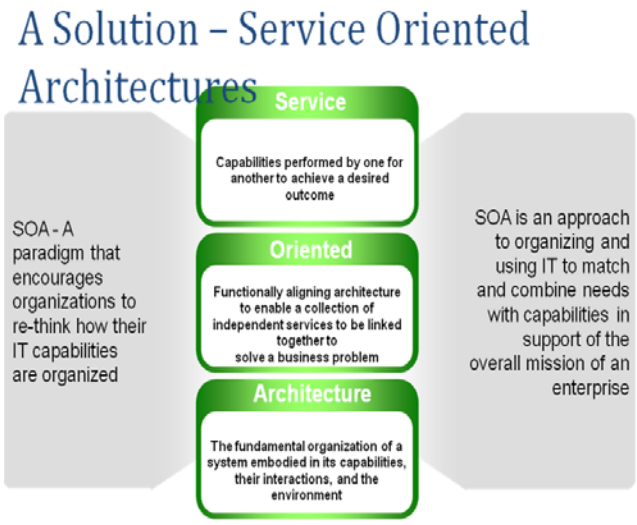

Figure 1: Service Oriented Architecture

Service Oriented Architecture (**SOA**) principles have been the foundation for the evolution of transactional systems to e-business and end-to-end business process integration. Basically Service Oriented Architecture (SOA) is a business-centric IT architectural approach that supports integrating your business as linked, repeatable business tasks, or services.

II. DEFINING SERVICE ORIENTED ARCHITECTURE GOVERNANCE

a. Governance:

There are two fundamental aspects of governance. The first aspect involves the processes established by an organization to determine who is empowered to make certain decisions. The second aspect includes the mechanisms and policies that are used by the organization to measure and control the way those decisions are implemented. Together, these aspects form a governance framework. Governance is the structure of relationships and processes to direct and control the SOA components in order to achieve the goal of the enterprises.[1]

b. IT Governance:

The processes that ensure the effective and efficient use of IT in enabling an organization to achieve its goals

c. Enterprise Architecture Governance:

A mechanism to ensure projects apply perspective guidance provided by the EA process

d. SOA Governance extends IT Governance:

SOA governance is an extension of IT governance, which is an extension of corporate governance.[2] Since SOA is a joint business/IT environment, SOA governance is an extension of IT governance to perform two functions:

(a). to define the decision rights for the new services within IT

(b). to define the new decision rights that now exist between the business and IT organizations.

The function of SOA Governance is primarily to:

(a). Establish decision rights for the development, deployment, operations and management of new services

(b). Monitor and report decisions and results for communicating governance results

As a specialization of IT governance, SOA governance suggests how IT governance's decision rights, policies,

procedures and measures need to be modified and augmented for successful SOA adoption.

Figure 2: SOA Governance

A. Scope of SOA Governance:

a. Delivering value to the stakeholders: investments are expected to return a benefit to the stakeholders-this is equally true for SOA.
b. Compliance to standards or laws: IT systems require auditing to prove their compliance to regulations like the Sarbanes-Oxley Act.
c. Change Management: Changing a service often has unforeseen consequences as the service consumers are unknown to the service providers. This makes an impact analysis for changing a service more difficult than usual.
d. Ensuring quality of services: The flexibility of SOA to add new services requires extra attention for the quality of these services. This concerns both the quality of design and the quality of services.

B. Purpose of SOA Governance:

SOA Governance begins with mapping corporate, business and IT policies to identify specific SOA business services. It then defines and enforces the compliance rules and policies for managing those services, and dictates policies for services reuse, IT, compliance and security.SOA Governance is only as strong as the adoption and use of clearly defined business requirements and processes by key stakeholders and user groups. At the core of Governance is the ability to monitor, measure, and analyze the organization's SOA service Network.[5]

C. SOA Governance benefit:

a. Agility: SOA governance can facilitate fast, effective decision making across both business and IT, and enhance the ability to rapidly build, configure and assemble services to form innovative solutions in the marketplace, reducing bureaucratic obstacles that get in the way

b. Speed to Market: SOA governance can speed resolution when things do not work according to the plan. People will understand who to go to and how best to resolve issues for maximum effectiveness. This knowledge can help speed change, enabling organizations to react more quickly and decisively to competitive threats and marketplace opportunities

c. Reduced Cost: Acceptance of and agreement on services that provide the greatest value encourages adoption and reuse of those services and reduces wasted effort and cost. Tracking and managing to standards helps guide users and developers to know what to do and when and where to look for available services. As existing service assets are leveraged across the organization, return on investment improves.

III. COMPARISON BETWEEN IT GOVERNANCE AND SOA GOVERNANCE

IT governance is, well, governance for IT; namely: The application of governance to an IT organization, its people, processes and information to guide the way those assets support the needs of the business. SOA governance is a specialization of IT governance that puts key IT governance decisions within the context of the lifecycle of service components, services, and business processes. It is the effective management of this lifecycle that is the key goal of SOA governance.

IT governance is broader than SOA governance. IT governance covers all aspects of IT, including issues that affect SOA like data models and security, as well as issues beyond SOA like data storage and desktop support. SOA governance addresses aspects of the service life cycle such as: planning, publishing, discovery, versioning, management, and security.

Governance becomes more important in SOA than in general IT. In SOA, service consumers and service providers run in different processes, are developed and managed by different departments, and require a lot of coordination to work together successfully. For SOA to succeed, multiple applications need to share common services, which means they need to coordinate on making those services common and reusable. These are governance issues, and they're much more complex than in the days of monolithic applications or even in the days of reusable code and components.[4]

As companies use SOA to better align IT with the business, they can ideally use SOA governance to improve overall IT governance. Employing SOA governance is a key if companies are to realize the benefits of SOA. For SOA to be successful, SOA business and technical governance is not optional, it is required.[4]

SOA governance builds on existing IT governance techniques and practices. A key aspect of IT governance when using object-oriented technologies like Java 2 Platform, Enterprise Edition (J2EE) is code reuse. Code reuse also illustrates the difficulties of IT governance.[4]

While IT principles are a related set of high-level statements about how IT should be used in the business, SOA principles define the general guiding principles to be followed while coming up with an enterprise SOA. The IT principles should be derived from a higher-level set of business principles that management owns. For example, the following is a sample list of business principles:

a. Standardize processes and technologies wherever possible.
b. Alignment and responsiveness to negotiated business principles.

The following could be derived from those IT principles:

a. Architectural integrity
b. Responsive, flexible, and extendible infrastructure
c. Rapid and efficient deployment of applications

The IT principles can be mapped to the business principles as follows: Architectural integrity (the first IT principle) provides for standardized processes and technologies (the first business principle) while rapid and efficient deployment of applications (the third IT principle) promotes alignment and responsiveness to negotiated usiness principles (the second business principle).

Some guiding SOA principles that drive the service model could be:

a. Compliance to standards that are industry-specific as well as cross organizational
b. Service identification and categorization
c. Service provisioning
d. Service monitoring and tracking
e. Capability of services to be composed in order to realize different business services

The SOA principles also influence the IT principles.[4]

IV. SOA GOVERNANCE ASPECTS

A. Service Definition:

The most fundamental aspect of SOA governance is overseeing the creation of services. Services must be identified, their functionality described, their behavior scoped, and their interfaces designed. The service's boundaries should encapsulate a reusable, context-free capability. The interface should expose what the service does, but hide how the service is implemented and allow for the implementation to change or for alternative implementations. When services are designed from scratch, they can be designed to model the business; when they wrap existing function, it can be more difficult to create and implement a good business interface.

An interesting example of the potential difficulties in defining service boundaries is where to set transactional boundaries. A service usually runs in its own transaction, making sure that its functionality either works completely or is rolled back entirely. However, a service coordinator (a.k.a. orchestrator or choreographer) may want to invoke multiple services in a single transaction (ideally through a specified interaction like WS-Atomic Transactions). This task requires the service interface to expose its transaction support so that it can participate in the caller's transaction. But such exposure requires trust in the caller and can be risky for the provider. For example, the provider may lock resources to perform the service, but if the caller never finishes the transaction (it fails to commit or roll back), the provider will have difficulty cleanly releasing the resource locks. As this scenario shows, the scope of a service and who has control is sometimes no easy decision. [3]

B. Service deployment life cycle:

Services don't come into being instantaneously and then exist forever. Like any software, they need to be planned, designed, implemented, deployed, maintained, and ultimately, decommissioned. The application life cycle can be public and affect many parts of an organization, but a service's life cycle can have even greater impact because multiple applications can depend on a single service. While there is no one-size-fits-all life cycle that is appropriate for all services and all organizations, a typical service development life cycle has five main stages:

a. ***Planned:*** A new service that is identified and is being designed, but has not yet been implemented or still being implemented.
b. ***Test:*** Once implemented, a service must be tested (more on testing in a moment). Some testing may need to be performed in production systems, which use the service as if it were active.
c. ***Active:*** This is the stage for a service available for use and what we typically think of as a service. It's a service, it's available, it really runs and really works, and it hasn't been decommissioned yet.
d. ***Deprecated:*** This stage describes a service which is still active, but won't be for much longer. It is a warning for consumers to stop using the service.
e. ***Sunsetted:*** This is the final stage of a service, one that is no longer being provided. Registries may want to keep a record of services that were once active, but are no longer available. This stage is inevitable, and yet frequently is not planned for by providers or consumers. [2]

One stage which may appear to be missing from this list is "maintenance." Maintenance occurs while a service is in the active state; it can move the service back into test to reconfirm proper functionality, although this can be a problem for existing users depending on an active service provider. Maintenance occurs in services much less than you might expect; maintenance of a service often involves not changing the existing service, but producing a new service version.

C. Service versioning:

No sooner than a service is made available, the users of those services start needing changes. Bugs need to be fixed, new functionality added, interfaces redesigned, and unneeded functionality removed. The service reflects the business, so as the business changes the service needs to change accordingly.

With existing users of the service, however, changes need to be made judiciously so as not to disrupt their successful operation. At the same time, the needs of existing users for stability cannot be allowed to impede the needs of users desiring additional functionality.

Service versioning meets these contradictory goals. It enables users satisfied with an existing service to continue using it unchanged, yet allows the service to evolve to meet the needs of users with new requirements. The current service interface and behavior is preserved as one version, while the newer service is introduced as another version. Version compatibility can enable a consumer expecting one version to invoke a different but compatible version.

While versioning helps solve these problems, it also introduces new ones, such as the need to migrate.[5]

D. Service migration:

Even with service versioning, a consumer cannot depend on a service -- or more specifically, a desired version of that service

-- to be available and supported forever. Eventually, the provider of a service is bound to stop providing it. Version compatibility can help delay this "day of reckoning" but won't eliminate it. Versioning does not obsolete the service development life cycle, but it enables the life cycle to play out over successive generations.

When a consumer starts using a service, it is creating a dependency on that service, a dependency that has to be managed. A management technique is for planned, periodic migration to newer versions of the service. This approach also enables the consumer to take advantage of additional features added to the service.

However, even in enterprises with the best governance, service providers cannot depend on consumer migration alone. For a variety of reasons -- legacy code, manpower, budget, priorities -- some consumers may not migrate in a timely fashion. Does that mean the provider must support the service version forever? Can the provider simply disable the service version one day after everyone should have already migrated? [5]

E. Service registries:

How do service providers make their services available and known? How do service consumers locate the services they want to invoke? These are the responsibilities of a service registry. It acts as a listing of the services available and the addresses for invoking them.

The service registry also helps coordinate versions of a service. Consumers and providers can specify which version they need or have, and the registry then makes sure to only enumerate the providers of the version desired by the consumer. The registry can manage version compatibility, tracking compatibility between versions, and enumerating the providers of a consumer's desired version or compatible versions. The registry can also support service states, like test and (as mentioned before) deprecated, and only make services with these states available to consumers that want them.

When a consumer starts using a service, a dependency on that service is created. While each consumer clearly knows which services it depends on, globally throughout an enterprise these dependencies can be difficult to detect, much less manage. Not only can a registry list services and providers, but it can also track dependencies between consumers and services. This tracking can help answer the age-old question: *Who's using this service?* A registry aware of dependencies can then notify consumers of changes in providers, such as when a service becoming deprecated.

IBMâ€™s WebSphere Service Registry and Repository is a product for implementing service registries. It acts as a repository for service definitions, and registry for providers of those services. It provides a centralized directory for developers to find the services available for reuse, as well as use at runtime for service consumers and enterprise service buses (ESBs) to find providers and the addresses for invoking them.[9]

F. Service message model:

In a service invocation, the consumer and provider must agree on the message formats. When separate development teams are designing the two parts, they can easily have difficulty finding agreement on common message formats. Multiply that by dozens of applications using a typical service and a typical application using dozens of services, and you can see how simply negotiating message formats can become a full-time task.

A common approach for avoiding message format chaos is to use a canonical data model. A canonical data model is a common set of data formats that is independent of any one application and shared by all applications. In this way, applications don't have to agree on message formats, they can simply agree to use existing canonical data formats. A canonical data model addresses the format of the data in the message, so you still need agreement around the rest of the message format -- such as header fields, what data the message payload contains, and how that data is arranged -- but the canonical data model goes a long way toward reaching agreement.

A central governance board can act as a neutral party to develop a canonical data model. As part of surveying the applications and designing the services, it can also design common data formats to be used in the service invocations.[5]

G. Service monitoring:

A composite application, one that combines multiple services, is only as reliable as the services it depends on. Since multiple composite applications can share a service, a single service failure can affect many applications. SLAs must be defined to describe the reliability and performance consumers can depend on. Service providers must be monitored to ensure that they're meeting their defined SLAs.

A related issue is problem determination. When a composite application stops working, why is that? It may be that the application head, the UI that the users interface with, has stopped running. But it can also be that the head is running fine, but some of the services it uses, or some of the services that those services use, are not running properly. Thus it's important to monitor not just how each application is running, but also how each service (as a collection of providers) and individual providers are also running. Correlation of events between services in a single business transaction is critical.

Such monitoring can help detect and prevent problems before they occur. It can detect load imbalances and outages, providing warning before they become critical, and can even attempt to correct problems automatically. It can measure usage over time to help predict services that are becoming more popular so that they can run with increased capacity.[8]

H. Service ownership:

When multiple composite applications use a service, which is responsible for that service? Is that person or organization responsible for all of them? One of them; if so, which one? Do others think they own the service? Welcome to the ambiguous world of service ownership.

Any shared resource is difficult to acquire and care for, whether it's a neighborhood park, a reusable Java framework, or a service provider. Yet a needed pooled resource provides value beyond any participant's cost: Think of a public road system.

Often an enterprise organizes its staff reporting structure and finances around business operations. To the extent that an SOA organizes the enterprise's IT around those same operations, the department responsible for certain operations can also be responsible for the development and run time of the IT for those operations. That department owns those services. Yet the services and composite applications in an SOA often don't follow an enterprise's strict hierarchical reporting and financial structure, creating gaps and overlap in IT responsibilities.

A related issue is user roles. Because a focus of SOA is to align IT and business, and another focus is enterprise reuse, many different people in an organization have a say in what the services will be, how they will work, and how they'll be used. These roles include business analyst, enterprise architect, software architect, software developer, and IT administrator. All of these roles have a stake in making sure the services serve the enterprise needs and work correctly.

An SOA should reflect its business. Usually this means changing the SOA to fit the business, but in cases like this, it may be necessary to change the business to match the SOA. When this is not possible, increased levels of cooperation are needed between multiple departments to share the burden of developing common services. This cooperation can be achieved by a cross-organizational standing committee that, in effect, owns the services and manages them. [7]

I. Service testing:

The service deployment life cycle includes the test stage, during which the team confirms that a service works properly before activating it.

SOA increases the opportunity to test functionality in isolation and increases the expectation that it works as intended. However, SOA also introduces the opportunity to retest the same functionality repeatedly by each new consumer who doesn't necessarily trust that the services it uses are consistently working properly. Meanwhile, because composite applications share services, a single buggy service can adversely affect a range of seemingly unrelated applications, magnifying the consequences of those programming mistakes.

To leverage the reuse benefits of SOA, service consumers and providers need to agree on an adequate level of testing of the providers and need to ensure that the testing is performed as agreed. Then a service consumer need only test its own functionality and its connections to the service, and can assume that the service works as advertised. [6]

J. Service security:

Security is a difficult but necessary proposition for any application. Functionality needs to be limited to authorized users and data needs to be protected from interception. By providing more access points to functionality (that is, services), SOA has the potential to greatly increase vulnerability in composite applications.

SOA creates services that are easily reusable, even by consumers who ought not to reuse them. Even among authorized users, not all users should have access to all data the service has access to. For example, a service for accessing bank accounts should only make a particular user's accounts available, even though the code also has access to other accounts for other users. Some consumers of a service have greater needs than other consumers of the same service for data confidentiality, integrity, and no repudiation.

Service invocation technologies must be able to provide all of these security capabilities .Access to services has to be controlled and limited to authorized consumers. User identity

Access to services has to be controlled and limited to authorized consumers. User identity must be propagated into services and used to authorize data access. Qualities of data protection have to be represented as policies within ranges. This enables consumers to express minimal levels of protection and maximum capabilities and to be matched with appropriate providers who may, in fact, include additional protections.[5]

V. DISCUSSION AND CONCLUSIONS

This paper reviewed the SOA Governance and its various aspects. SOA governance has many aspects, such as: Service definition (the scope, interface, and boundaries of a service),Service deployment lifecycle (the lifecycle stages),Service versioning (including compatibility),Service migration (deprecation and sunsetting),Service registries (dependencies),Service message model (canonical data models),Service monitoring (problem determination),Service ownership (corporate organization),Service testing (duplicated testing),Service security (including ranges of acceptable protection).This paper also addressed the importance of implementing an effective SOA and IT governance in any enterprise which considers IT to be one of its key assets to generating revenue and staying competitive in the market. Governance is about creating a system of incentives and penalties to influence the right behavior. The best way to incentivize the organization to meet SOA Governance objectives is to establish formal goals, by which each IT group will be evaluated throughout the year.

VI. REFERENCES

[1]. Michael Niemann, “Governance for SOA:An Implementation Approach”, CEUR workshop proceedings vol-374,March 2008,p 5, ,ISSN 16/3-0073.

[2]. William A. Brown,Garry Moore,William Tegan,”SOA Governance-IBM’s approach” , **White paper**, August 2006.

[3]. David J.N. Artus, "SOA realization: Service design principles," IBM developerWorks; February 2006 .

[4]. Tilak Mitra, "A case for SOA governance," IBM developerWorks; August 2005.

[5]. Tony Cowan,"Services security with WebSphere Application Server V6, Part 1: Introduction to security architectures," IBM developerWorks; April 2006.

[6]. Bobby Woolf,"SOA development using service mocks," IBM developerWorks; December 2005.

[7]. Mandy Chessell and Birgit Schmidt-Wesche,"SOA programming model for implementing Web services, Part 10: SOA user roles," IBM developerWorks; February 2006.

[8]. Wilfred Jamison and Richard Duggan,"Monitor business IT services using IBM Tivoli Monitoring for Transaction Performance," IBM developerWorks; June 2005.

[9]. Jenny Ang, Luba Cherbakov, and Mamdouh Ibrahim, "SOA antipatterns: The obstacles to the adoption and successful realization of Service-Oriented Architecture," IBM developerWorks; November 2005

[10]. Mike P.Papazoglou and Willem-jan van den Heuvel," Service Oriented Architectures:approaches, technologies and research issues",The VLDB Journal 16,2007,p389-415,doi 10.1007/s00778-007-0044-3