zapthink white paper

WHY SOA GOVERNANCE MUST START AT DEVELOPMENT TIME

ZapThink, LLC • 108 Woodlawn Road • Baltimore, MD 21210 • info@zapthink.com • www.zapthink.com

WHY SOA GOVERNANCE MUST START AT DEVELOPMENT TIME

September 2007

Analyst: David Linthicum

Abstract

Now that SOA is moving from the planning to the project levels, there is a clear need to manage SOA's assets, as well as maximize its key values: Agility and reuse. Thus, the use of a well-defined and well-implemented SOA governance system is critical to the success of SOA. Choosing the right governance system requires that you carefully consider specific attributes of the enterprise domain, including an inventory of the major SOA resources such as data, Services, and processes, as well as interactions and dependencies, and how all relate to the notions of policy management, service agreements, and security. Clearly, these interrelationships are so complex and far reaching that a SOA governance system is an absolute necessity.

So, how do you select and implement a governance system? There are a few key things to consider as you define and build your SOA, and clear steps you must follow to achieve success. In this paper we'll take a look at the concept of SOA governance, as well as the steps needed to implement governance within your SOA problem domain.

I. Example of a Business Process Model Leading to Service Prioritization

At the genesis of every SOA's development cycle is the Business Process Model. We use this model as a top-down approach to SOA, decomposing the core business processes down to Services, and then we define the interaction of those Services, including interfaces, semantics, policy, security, etc.

It's within this model that we perform some rudimentary analysis. First and foremost is the identification of core Services, which we'll call candidate Services. The artifact for this is the metaservices directory, or the information about Services which is both stored and managed to create a repository for the lifecycle of the Service.

Next we focus on *Service prioritization*, or the notion of placing Services in priority order, from the least to the most important. The notion here is to track Services in order of their expected usage, and thus provide a better Service management approach, including an understanding of which Services will need higher-level SLAs and thus, perhaps, high availability mechanisms.

Service implementation is the process of designing and building the Service. This includes Service modeling and design, development of the Services, testing the Services, and preparing them for deployment.

Other activities that need to occur include:

- Service Deployment
- Service Consumption
- Service Versioning
- Service Impact Analysis
- Service Configuration Management
- Service Policy Management

II. Why "Development Time" Governance is a Necessity

Development time SOA governance works up from the data to the Services, gathering key information as it goes.

In essence, development time SOA governance works up from the data to the Services, gathering key information as it goes. Thus, you typically begin by defining the underlying data schema and turning that into metadata, and perhaps an abstraction of the data. Then, working up from there, you further define the Services that interact with the data, data Services, and then transactional Services on top of that. You can further define that into processes or orchestration. All this occurs with development time information managed within the development time SOA governance system.

Development Time SOA Governance, as the name implies, typically provides an integrated registry/repository that attempts to manage a Service from its inception through development to operational deployment, but typically not during run time execution of the Service.

Key components of development time SOA governance include:

- A registry and/or repository for the tracking of Service development, management, policy, security, and testing artifacts. To be most effective, this registry/repository should be integrated with the other development time SOA governance tools listed below.
- Development tools, including Service modeling, code development (typically within an IDE such as Eclipse or Microsoft Visual Studio), dependency tracking, policy creation and management, and other tools that assist in the design of Services.
- Deployment tools, including Service deployment, typically through binding with external development environments.
- Testing tools and Services, providing the developer/designer with the ability to create a test plan and testing scenarios, and then leveraging Service testing technology.

III. Understanding the Journey

So, now that we understand the notion of SOA development time governance, it's time to backup and understand the larger, more holistic issues, including:

- Portfolio Management
- SDLC Management
- Processes
- Tools
- People

Portfolio Management

IT portfolio management is the application of systematic management to large classes of items managed by enterprise Information Technology (IT) capabilities. Examples of IT portfolios would be planned initiatives, projects, and ongoing IT Services (such as application support). The promise of IT portfolio management is the quantification of previously mysterious IT efforts, enabling measurement and objective evaluation of investment scenarios.

The promise of IT portfolio management is the quantification of IT efforts, enabling measurement and objective evaluation of investment scenarios.

In the world of SOA, this becomes even more complex considering the number of Services (not to mention the underlying elements – components, schemas, adapters, data views, mainframe APIs, etc. – that make up the implementation of these Services) that are entered into the equation, and thus we have the need to track these assets using a SOA governance solution. Management of this portfolio focuses on proposed spending against established systems in relation to their value. These systems may be assessed as to their contribution to corporate profitability, and also on non-financial criteria such as stability, usability, and technical obsolescence. Thus, there is a need to capture the information about each Service, and abstraction of that Service into composites or processes, or both. From there we have a clear understanding of how that Service, and collections of Services, are able to add to the bottom line of the business, or how they perform within our portfolio.

The enabling technology behind portfolio management is the ability to provide a location to store information about the assets. In the case of SOA: Services and information. This requires a complete understanding of the Services and data, including owners, purpose, policy, design, development, applications semantics, etc.. This becomes core to both the understanding and success of the SOA, including the ability to track the asset and its benefit to the organization.

SDLC Management

The Software Development Lifecycle (SDLC), as applied to SOA, ensures that Services and applications are developed properly, and that the releases and changes are managed properly. Services, in the world of SOA, can be thought of as small applications. However, they are small applications that are linked to many larger applications (composites or processes), and thus have far reaching effects within the existing IT infrastructure.

Thus, there needs to be both a process and a set of mechanisms that lead the Service designer and developer through a set of pre-defined steps that link directly with a design-time governance technology that can track the software asset through requirements, development, testing, deployment, and then finally change management through the process of maintenance.

Processes

In order to create effective SOA, there need to be certain processes that are established, and followed, throughout design, development, testing, and deployment. These are processes that assure that the Services created, and thus the resulting composite and bound processes, are ready for mission-critical operations.

Tools

The use of tools within the Service development process (SDLC) provide the designer, developer, and the architect with a place to create and store artifacts, as well as track critical information, such as metadata, Service information, process information, and how all of these assets interact and relate. In the world of SOA, the ability to track these Services assets is very important considering the complexity of the architecture.

People

Finally, the people need to understand both the processes and supporting tools. This includes the steps for designing, developing, and deploying Services. Moreover, how all of these Services inter-relate and form the final architecture, and how people are points of processing within the system, as well as Services.

IV. SOA requires an Upfront Prioritization Strategy

When building SOA, the first step is to create a core prioritization strategy for the assets that are to be created. This means figuring out which Services should be designed and built before other Services, in which sequence, and thus the priorities of each asset. We address this through the notions of release and change management, configuration management, and performance management.

Release and Change Management

What's needed within the world of SOA is a dedicated resource to oversee the integration and flow of development, testing, deployment, and support of Services. Thus, we have the notion of release and change management. What's important here is that this occurs, and does so in such a way that the release of the Services, and perhaps composites and processes, go through a pre-defined process where the assets are tracked using a centralized repository and/or registry.

Configuration Management

Configuration management is the discipline of keeping evolving Services under control.

Configuration management is the control and adaptation of the evolution of complex systems, such as SOA. Or, in other words, it is the discipline of keeping evolving Services under control, and thus contributes to satisfying quality constraints. Configuration management, as it works with SOA, concerns the storage of the entities or assets produced during the SDLC. Configuration management, using a repository and/or registry, is able to track the use of these assets or entities, allowing the architect or SOA manager to monitor and control change, and therefore have a clear understanding of the impact of change.

Performance Management

Performance management refers to the discipline within systems management that focuses on monitoring and managing the performance and availability of Services. Thus the ability to detect, diagnose, remedy and report on Service performance issues to ensure that Service performance meets or exceeds the requirements of SOA-based applications consuming those Services.

V. Taking a "Lifecycle Approach"

SOA has the macro lifecycle of the architecture, in general, and the individual lifecycles of the Services, processes, composites, and other components.

SOA, while it is a complex architecture, has a lifecycle: The macro lifecycle of the architecture, in general, and the individual lifecycles of the Services, processes, composites, and other components. Therefore, we must learn to manage the lifecycles and track the assets throughout, and thus the architecture as a whole.

There are a few things to think about here:

- The overall lifecycle of the architecture, and how it relates to the underlying components.
- The lifecycle of the Services.
- The lifecycle of the composites.
- The lifecycle of the processes.
- Relationships and dependencies between all assets.

The use of a lifecycle denotes that we are taking each asset through a complete process, making sure to consider the design of the asset or Service, or the creation of the Service design when considering the requirements. This guarantees that there is no wasted work in the development stage, insuring that the Service meets the particular needs of the problem domain.

From requirement and design we move into the development stage, or the actual creation of the Service or other asset. After development the Service should undergo testing, and then go through a deployment process, as it's implemented within the SOA. Moreover, a rigorous change management process needs to be put into place around the Service, as well as performance management to assure that the Service is living up to the expectations of the architecture.

The use of the lifecycle approach requires both an understanding of the process as well as discipline, and the correct use of enabling technology such as design-time SOA governance, leveraging a well designed repository to track each asset/Service through the lifecycle. In the world of SOA, the role of the SOA governance infrastructure is more critical, considering the complexity and far-reaching nature of the architecture, and the fact that, while there is a holistic lifecycle of the architecture in general, there are many smaller lifecycles, such as Service lifecycles, that make up that architecture.

VI. Migrating to SOA

Considering what we've stated so far, the migration to SOA is one that, while requiring proper attention to process and management, pays large dividends to the IT organization in terms of efficiency and flexibility. To that end there are a few steps, or processes, that one most go through to get to SOA from an existing environment. They include:

- A firm understanding of and access to legacy assets
- Mid-level architectural governance
- Effective coordination between Service producers and consumers

A Firm Understanding of and Access to Legacy Assets

Part of the migration process is to understand what you currently have in place (i.e., manage the bottom-up nature of SOA in concert with the top-down architectural perspective). In most enterprises, candidate Services exist within legacy assets, and leverage all sorts of programming languages and enabling technology.

As candidate Services are identified, information about those assets should be stored in a common repository.

Candidate Services are entities of both behavior and information bound to behavior that exists within existing systems. As these assets are identified, information about those assets should be stored in a common repository. From there you can identify which assets will be used to support development of the prioritized Services identified by portfolio management processes, place each prioritized Service within its own lifecycle, and track the maturation of these Services (and their dependencies on legacy systems and assets) through Service enablement/development, deployment, and change management.

Mid-level Architectural Governance

The core problem with a loosely-coupled architecture, such as SOA, is that portfolio decisions made at the business level get lost in the project-specific activities of IT teams as they build out Services and applications.

Mid-level architectural governance ties business priorities and portfolio-level decisions to project-specific Service development and consumption activities. The core problem with a loosely-coupled architecture, such as SOA, is that portfolio decisions made at the business level get lost in the project-specific activities of IT teams as they build out Services and applications. Project teams, left to their own devices, will optimize for their own specific needs. This typically leads to narrowly focused Services that are:

- Not flexible enough to reuse across multiple business processes.
- Redundant with other Services built out by other projects but not communicated across the organization because of the project-focused nature of development teams.

Mid-level architecture governance provides the necessary oversight to ensure that broader business needs and objectives are kept in front of IT project teams (e.g., by reviewing candidate Services to identify other prioritized business processes that need similar functionality and broadening the Service interface definition to support those other processes up front).

Effective Coordination between Service Producers and Consumers

Finally, there needs to be effective coordination between Service producers and consumers, both during development time and run time. During development time, application developers must have ready access to both available and planned Services as they build their orchestrations and other client-side application components. This means exposing assets managed by the development time repository/registry directly in the IDE, the application developer's tool of choice. During run time, decisions made at development time need to be reflected into the SOA run time management and monitoring environments such that systems used to stand up Services for consumption are talking to the systems that need to consume those Services as part of the business solution. There are a few things that occur here, including:

- Leveraging discovery to locate produced Services.
- Finding and validating the produced Services, both from the business (e.g., should this application be allowed to use this Service) and technical (e.g., is this Service capable of providing an SLA that meets the needs of the consuming application) perspectives.
- Resolving protocol and semantic issues.
- Facilitating communications across the various stakeholders in both the architecture/design-time and run time domains.

VII. The ZapThink Take

What's core to the concepts presented in this paper is the importance of understanding your own issues, in your own domain, before attempting to design and develop the components of your SOA. In essence, you need to have an information level, Service level, and process level understanding, and also consider the concepts of prioritization, discovery, and lifecycle management of the SOA components.

There is a clear need for development time SOA governance, and the ability to track all key assets of the SOA throughout their lifecycle, including requirements, design, development, and change management.

The fact is, all of these activities are critical to the success of the SOA, but are daunting when you consider manually attempting these tasks. Thus, there is a clear need for development time SOA governance, and the ability to track all key assets of the SOA throughout their lifecycle, including requirements, design, development, and change management going forward. Indeed, without this key enabling technology the SOA would likely grow too complex to build, not to mention change, as business needs change.

If the core value of SOA is agility, then the ability to track all of the components of SOA, and change them as needed with a clear understanding of all of the interdependencies and patterns, is a critical success factor. Considering all of that, development time SOA governance is critical to the success as well.

VIII. Understanding LogicLibrary

LogicLibrary is currently the leading provider of development time SOA governance, and their Logidex product provides a good example of a comprehensive SOA governance system. LogicLibrary's Logidex is a development time integrated registry/repository with configurable governance policies that allow organizations to actively track and manage Services from inception through deployment. Logidex provides a combination of a repository for Services production, a registry for Services and artifact consumption, and integrations with leading Service repositories such as HP SOA Systinet and IBM WSRR to provide end-to-end governance capabilities for the SOA lifecycle.

LogicLibrary’s Logidex addresses the needs of development time governance. In addition, through partnerships and integrations with the leading Service registries and repositories, Logidex provides a unique approach to end-to-end SOA lifecycle governance.

Copyright, Trademark Notice, and Statement of Opinion

About ZapThink, LLC

ZapThink is an IT advisory and analysis firm that provides trusted advice and critical insight into the architectural and organizational changes brought about by the movement Service Orientation and Enterprise Web 2.0. We provide our three target audiences of IT vendors, service providers and end-users a clear roadmap for standards-based, loosely coupled distributed computing – a vision of IT meeting the needs of the agile business.

ZapThink helps its customers in three ways: by helping companies understand IT products and services in the context of Service-Oriented Architecture (SOA) and the vision of Service Orientation, by providing guidance into emerging best practices for Web Services and SOA adoption, and by bringing together all our audiences into a network that provides business value and expertise to each member of the network.

ZapThink provides market intelligence to IT vendors and professional services firms that offer XML and Web Services-based products and services in order to help them understand their competitive landscape, plan their product roadmaps, and communicate their value proposition to their customers within the context of Service Orientation.

ZapThink provides guidance and expertise to professional services firms to help them grow and innovate their services as well as promote their capabilities to end-users and vendors looking to grow their businesses.

ZapThink also provides implementation intelligence to IT users who are seeking guidance and clarity into the best practices for planning and implementing SOA, including how to assemble the available products and services into a coherent plan.

ZapThink's senior analysts are widely regarded as the "go to analysts" for SOA and Enterprise Web 2.0 by vendors, end-users, and the press. Respected for their candid, insightful opinions, they are in great demand as speakers, and have presented at conferences and industry events around the world. They are among the most quoted industry analysts in the IT industry. ZapThink was founded in November 2000 and is headquartered in Baltimore, Maryland.

ZAPTHINK CONTACT:
ZapThink, LLC
108 Woodlawn Road
Baltimore, MD 21210
Phone: +1 (781) 207 0203
Fax: +1 (815) 301 3171
info@zapthink.com