Service Governance and Virtualization For SOA

Frank Cohen
Email: fcohen@pushtotest.com

Brian Bartel
Email: bbartel@pushtotest.com

November 7, 2006

PushToTest

Table of Contents

Introduction **3**

Design-Time Software Development Governance For Increased Reuse 4

Run-time Governance For Increased Control and Flexibility 4

What Is Service Virtualization? **5**

Service Virtualization in Action **7**

Part 1 – Service Development 7

Part 2 – Service Reuse 8

Service Virtualization in Operations 9

Summary and Action List **10**

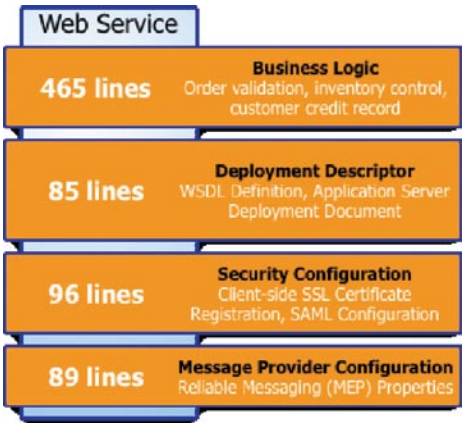
Resources **11**

Introduction

Many attempts over the past forty years of software development have focused on improving the speed of development, improving quality by lowering bug counts, and making software easier to maintain over time by building in flexibility. The tools and techniques we use today evolved from centralized mainframe systems to today's environment where n-tier architectures use Web servers, application servers and Enterprise Service Buses (ESB). Best practices expressed as patterns emerged and have become widely adopted as developers moved from structured programming to object oriented programming to Service Oriented Architecture (SOA). All of this has helped speed up development and lower IT costs for enterprises.

Unfortunately these evolutions have also added complexity. For instance, many developers today must add additional layers of code to core application logic just to deploy their application in a service environment. In our conversations with software developers we find too much effort going into coding for the communication, deployment and reuse of the service. One customer recently noted they were spending as much as 40% of their total effort managing this complexity. There is too much effort coding bindings, URLs, security policy, and trying to keep in one's mind all the moving pieces of the average SOA application. This is negatively impacting IT agility and time-to-market.

Just look at the lines of code needed to implement a secure and production worthy SOAP-based Web Service today! We built a Web Service to sell licenses to our manuals. The business logic took 465 lines of Java code. However, we had to write another 270 lines of configuration, deployment, and property files to install the service in our application server, establish security for the service, and configure a reliable message provider for the service. Taken together 37% of the code exists simply to deploy our service.



Deployment and Configuration makes up 37% of the lines of code in our service.

We see this complexity today in two sets of technical challenges: *the lack of design-time software development governance* and *the lack of run-time governance*.

DESIGN-TIME SOFTWARE DEVELOPMENT GOVERNANCE FOR INCREASED REUSE

One typical complaint we hear from software developers and CIOs is that management treats each new software development project as a task that can be delivered through brute-force, if necessary. Even in SOA, as projects expand, services are not necessarily being reused. Instead of emphasizing reuse, this attitude institutionalizes an approach to software development that emphasizes one-off and developer-intensive efforts. The result is duplicated effort, added risk, burnout of critical developer human resources, and increasing complexity that makes it harder to maintain and change systems in the future.

"Establishing a Repository and Registry for elements used with the ESB was the most widely recognized stumbling block (73%) by best-in-class adopters of SOA."

Source: Aberdeen ESB and SOA Middleware Study, June 2006

Developers also have many choices to make at design-time that impact operations. For instance, developers have a choice of XML parsing libraries, communication protocols and message encoding styles, security mechanisms, and architectural patterns. Design-time governance is more than CVS, Maven, and other code repositories are able to give. Developers need a way to write business logic and deployment logic in reusable software code and to categorize and make the existing code easily available in registries and repositories to other software developers at design-time as services.

RUN-TIME GOVERNANCE FOR INCREASED CONTROL AND FLEXIBILITY

One result of these development realities is added complexity in IT operations. Software built on a one-off basis with hard-coded security, policy and other attributes *bakes* degraded quality into operations. It eventually makes the system brittle and prone to outages. When those outages occur, it is hard to figure out which individual component on which technology is the root cause. We often see IT resort to the server reset button or restoring to a backed up state as the best solution, regardless of the end-user experience.

"A lack of governance was the most widely recognized stumbling block for SOA adoption at 43% of all those surveyed."

Source: Aberdeen Group, June 2006

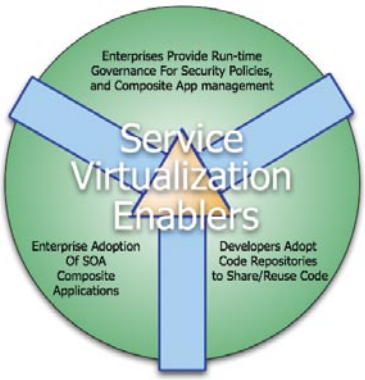
The majority of CIOs tell us that SOA fails most-often when there is an absence of run-time governance automation and manual techniques are in use. One current best practice for deploying, monitoring, and managing multiple interdependent services is to establish policies that can be enforced across the different technologies instead of coding these policies directly into the services. Replacing code with rules-based policies that can be configured at runtime independently of the core service code adds flexibility and improves reuse. Applying the same policy across services also makes administrators more productive and provides greater control over the environment. Some policies, such as security, have standards like WS-Security that are being adopted. Other types of policies, such as for enforcing service levels, are only just now starting to be possible. Previously they have been implemented in code or using content-based routing. Both are far from ideal.

Governance is a great start, but it doesn't completely solve the problems customers face around service reuse and simplified deployment and management across different service technologies. From time to time the software industry develops ways to simplify and remove some of the complexity that builds up over the years, in this case the complexity governance is helping to manage. For instance, application servers introduced *container-managed services* to move configuration code into deployment descriptors. Now we see a new category of software emerging to address these design-time and run-time governance challenges and get developer and administrator focus back on building the service logic. This is best described as *service virtualization*.

What Is Service Virtualization?

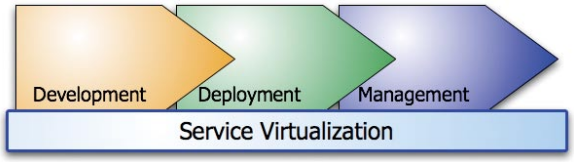
Service virtualization – originally coined by Daryl Plummer of Gartner in 2005 – is an abstraction layer that separates the service invocation from the request by a consumer. The idea is that you can focus on just the business logic for consumers and services, and deploy the code on a runtime framework that takes care of the rest. Service virtualization provides two major benefits. The first benefit is code reduction. A developer can implement a class that implements just the service functionality and literally deploy the class as a "virtualized" service in a service machine, or container, that makes all the other details configurable. The second benefit is greater reuse and flexibility. Service bindings, security, policy and other attributes can be reconfigured and changed as needed when reusing a service instead of having to change code. Ideally the infrastructure makes this easy.

IT adopted SOA first as a technique for building business applications as services composed together using orchestration technology. Over the last few years SOA has expanded to include event-driven principles – we now often hear this described by software architects as event-driven SOA. More recently enterprises have been adopting governance technologies – registries, repositories, service and policy management. Service virtualization builds on these technologies and optimizes them together as a natural *next-step*.



The SOA foundation choices that enable service virtualization.

Service virtualization takes a service container approach to software development. The software developer writes core service logic or a client that calls a service and the service virtualization framework takes care of deploying the services and consumers to service containers. Developers of the services or clients don't need to worry about the service location, the communication protocol, security requirements or related usage policies. It's all configured in the service container, outside of the code.



Service virtualization enables developers to write just the core business logic as a service or consumer and wrap it in a service container for deployment and management.

Enterprises adopting service virtualization benefit in several ways:

- Developers are more productive because they can focus on writing just the core business logic of the service and let the virtualization technology manage everything else. The best solution allows the developers to focus on their development environment of choice, and not have to use multiple environments and tools.
- Services are more easily promoted, reused and changed across departments because security, policies and communication over Web services or messaging is now configured, not coded into the service.

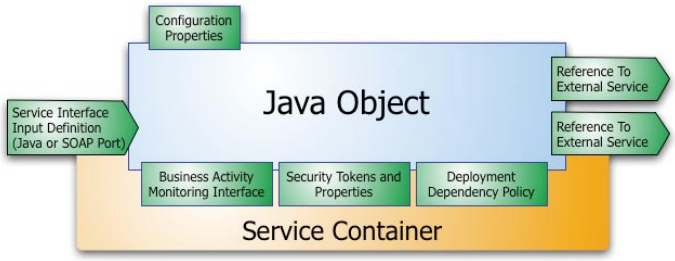
- Administrators are more productive because service and policy configuration, deployment and management are done centrally, not separately for each application server, ESB or orchestration engine.

Service Virtualization in Action

Consider the case of a large logistics company that wants to offer a new service to notify customers about package delivery delays. The notification would happen as the package moves through the system. An event-driven SOA is needed to provide the framework for the new service.

PART 1 – SERVICE DEVELOPMENT

The company has a logistics application for tracking packages as they move through trucks, airports and planes. The system is capable of emitting event-driven messages using standard SOAP over JMS. When employees wish to track a package they subscribe to events and are notified as the package moves through scanners in the airline system.



A service virtualization environment makes development easier by hosting core service logic and making deployment, communication and management rules-based configuration parameters.

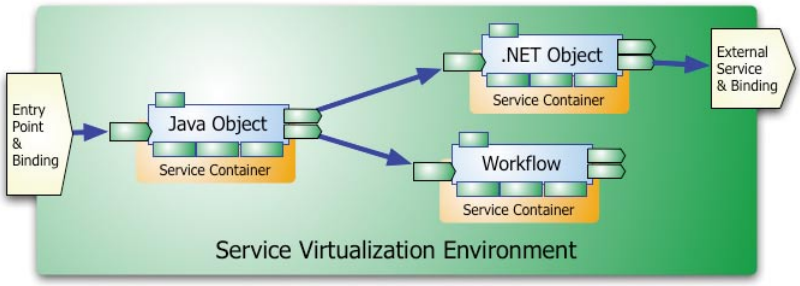
To establish the package tracking service a software developer writes a Java class to receive package shipment event notices using typical development environment tools (code editor, compiler and debugger.) The service virtualization technology and related tools generate the service container class around the service class, expose the configuration properties for the object, the service input definition for the interface for consumers to call the service, and a way for external services, human tasks, workflows, business rules, datasources, and connectors to invoke the service. The service virtualization tools package the Java class to be deployed in a Java runtime environment. The tools also make transactions, communication over HTTP or in this case a JMS-based messaging system, security and policies configurable. Instead of having to change code, developers and administrators only have to reconfigure these properties to reuse the service.

The container then manages these properties at runtime and allows centralized service deployment, communication, monitoring, policy and service management, and administration across different containers. The container is not the physical runtime, rather a part of each runtime that hosts each service. The Java class technically is running on the JVM, or in the case of an EJB on a Java EE-based application server. But the service is being invoked as part of the service container class and interacting through the service container with other services.

Scans of the shipping label as the package moves from handler to handler generate event messages that are broadcast to listening services. Our service keeps track of the package and an employee may check package delivery status through standard Web and service interfaces.

PART 2 – SERVICE REUSE

The company now makes a decision to allow customers to register to receive notifications of the forecasted delivery time as the package moves through the supply chain and also to enforce service level agreements. The logistics company uses the virtualized service from Part 1, adds new business logic to recognize when a package might arrive past its committed delivery time, and adds processes to try and maintain service levels. It also adds logic to send email notifications directly to customers as the package moves through the system.



Service virtualization enables a process modeler to assemble a set of service components, resolve reference dependencies and apply policies.

Scans of a package label as the package moves from handler to handler generate event messages. The software developer writes rules in a listening service to identify when a package is in danger of not being delivered on-time. A new orchestration using a Business Process Execution Language (BPEL) workflow engine compensates and maintains the service level agreement. This orchestration invokes other services, and once complete sends an email notification to the customer that includes a revised estimate of the delivery time.

The developer packages up the new services and BPEL workflow as a single application. The BPEL gets deployed in its own service container. An administrator defines a security policy and applies it to the BPEL orchestration and related services as part of the deployment process. This reuse continues. The company then uses these same Java services and BPEL to drive a .NET-based employee scheduling system to schedule expedited deliveries that help meet service commitments.

SERVICE VIRTUALIZATION IN OPERATIONS

Service virtualization delivers a common framework for deployment and administration within an abstract framework for extensibility. With service virtualization IT managers have a dynamic way to scale and deploy on-demand services, including service deployment to staging and production environments, service deployment simulation, and grid-like functions to move services from one application/datacenter cluster to another. For instance, service virtualization containers abstract the communication between containers, including communication over an ESB. The service need not even know about an ESB.

The service virtualization environment deploys objects that connect to an ESB to communicate with services inside or outside of the base network.

In our conversations with IT managers, today's new service development projects deliver an archive of services that need to be deployed together. At a minimum the developer delivers deployment instructions in the form of read-me documents and deployment descriptor files. Most also include at least partially automated deployment scripts. Service virtualization archives the services together as a single deployable unit that is managed through a central console. Two new standards efforts are making this possible.

- Service Component Architecture (SCA) uses code annotations – whether it is in Java, Java EE, or BPEL – to describe a service's functions, dependencies on other services, bindings, and other properties. Additionally, SCA also defines the service's packaging.

- Java Business Integration (JBI, JSR 208) is a pluggable architecture for a container that hosts service producer and consumer components. JBI enables service containers to plug into a messaging runtime to handle common deployment and service communication and mediation.

While SCA and JBI overlap in the way services are packaged, they are generally complementary and the overlap is likely to disappear over time. SCA and JBI are important for developers in that they enable service virtualization with standards-based containers and policy and management capabilities that simplify deployment.

While IT managers install and configure each container separately, the actual service deployment and management can be centralized. An administrator manages services, configures security, policy, communication and other elements centrally and quickly.

Service virtualization also makes all the metadata defined in development available in operations. This allows administrators to monitor and manage with a top-down, process-centric view of the applications as a whole instead of managing each of the services independently. This also allows administrators to automate impact analysis and dependency tracking based on the metadata.

Summary and Action List

We recommend that anyone evaluating tools for SOA development investigate a service virtualization framework, something we believe will become a standard part of computing. At a minimum we recommend you do the following:

1. Put in place an architecture that allows you to separate your core service logic from the rest of the code. This enables core service logic to be repurposed and reused by using a layering approach to service development.

2. Follow key standards and plan ahead. Learn more about SCA and JBI, as well as WS-Security and WS-Policy.

3. Put in place a clear roadmap for implementing design-time governance with a registry and repository, and runtime governance with policy and service management now. These are critical for larger SOA deployments and for getting your return on SOA.

4. Plan ahead for service virtualization to make sure you can adopt it. Under-stand and outline how each of your service runtimes become service containers, and how security, service and policy management, deployment and administration evolve into a common set services across your containers.

Resources

Articles on service virtualization are found in the following.

1. Software Component Architecture (SCA) is at http://www.osoa.org
2. Software Architectures Will Evolve From SOA and Events to Service Virtualization by Daryl Plummer at http://www.gartner.com

PushToTest is a software publishing and global services company specializing in business optimization and information systems test automation. Our expertise is put to work for enterprises needing to understand and solve scalability and performance problems in complex information systems, especially Web Services. PushToTest provides open-source SOA test automation tools for software developers, information architects, and CIOs. Details are found at **http://www.pushtotest.com**.