Service Virtualization: Managing Change in a Service-Oriented Architecture

Abstract

Load balancers, name servers (for example, Domain Name System [DNS]), and stock brokerage services are examples of virtual services in the areas of networking and financial services. In the area of service-oriented architecture (SOA), virtual services and their supporting abstraction layers are crucial to addressing the operational, integration, security, and lifecycle problems that delay and derail SOA deployment and success. This document introduces the concept of virtual services for SOAs and provides details about the characteristics of virtual services that enable SOA success.

SOA and Web services represent a significant opportunity to develop business systems that can adapt to business requirements in a timely and effective manner. However, operational characteristics of real SOA deployments can derail much of the expected return on investment.

Service virtualization addresses the critical operational, integration, and lifecycle challenges that can derail or delay SOA implementations. Without addressing these challenges through service virtualization and its supporting abstraction layer, many of the benefits of SOA are constrained at best, and at worst may not be achieved at all.

This document is part of a series describing policy, deployment, and operational concerns and solutions associated with SOA and Web services.

Virtualization and Abstraction

Virtual services abound in the real world, although often we are so familiar with them that we do not recognize them as such.

Stock Brokerage Services

In the world of stock trading, many different stock exchanges exist, but few people are aware of them, and even fewer are in a position to trade on these exchanges directly. Instead, most people use a virtual service known as a stock broker, who handles requests to buy or sell stock in companies irrespective of what stock exchange a company is listed on. People rely on the stock broker to implement the rules of the exchanges and to correctly interact with the interfaces for making trades.

People expect stock brokerage services to provide added value: for instance, by aggregating transactions and monitoring transactions for completeness and for tax-reporting purposes. Stock brokers are expected to know when changes occur such as the movement of a company stock listing from one board or exchange to another, and to analyze the flow of trades to make recommendations about the likely direction of future trading.

In this analogy, stock exchanges provide concrete or implemented services, the broker provides a virtual service, and the additional value that users derive is provided by the abstraction layer provided by the stock broker.

Networking Services

Common virtual service examples in the networking world include services such as name servers and load balancers.

A name server (for example, DNS) allows a network node to maintain a stable known name by which it is accessed even if its network address changes. Changes to the address or location of the node are concealed from applications that use the DNS name to access a network node. Application developers do not need to understand how the network infrastructure works and do not need to include additional complexity in their code to handle exception events or provide configuration and management. The virtual service is the exposed or published DNS name of the network node, and the implemented service is the physical node being accessed. The abstraction layer provided by the DNS includes value-added capabilities such as caching to reduce the cost of name translations.

Similarly, a load balancer exposes a single virtual instance of a server identity while supporting multiple instances of the implemented server. The load balancer abstraction layer then provides additional value by distributing the load for the virtual service in various ways across implemented services or by providing fault detection and failover to handle loss of an implemented service.

In each of these cases, there is a virtual or exposed service that people interact with (generally through a publication or discovery process), and then instances in various forms of the actual implemented service that supplies the physical service. In addition, in each case, an abstraction layer provides the logic that handles change or adds value on top of the originally exposed service. The naming service abstraction layer makes and distributes changes in addresses; the load balancer abstraction layer handles load distribution, failover, and fault detection.

What Is a Virtual Service?

The concept of a virtual service is simple but powerful. When a service is implemented, a service contract is created defining attributes that describe how a service consumer should interact with the service interface.[1] Rather than directly publish this implemented service interface with its service contract, an alternate service interface with a different address is published and deployed. This virtual service interface may define a different collection of attributes defining a new (but closely related) service contract that defines unique aspects of the use requirements of the service instance. An abstraction layer operates between the virtual service interface and the implemented service interface to harmonize differences between the interfaces, perform transformations as required, integrate with other infrastructures, and allow policy-based changes to be made.

Minimally, this approach provides an opportunity to change the address of the implemented service without affecting its dependant consumers. However, the supporting abstraction layer provides a much richer set of opportunities, not the least of which are policy controls that change or modify the capabilities of the published interface without requiring recoding of the service implementation. Different security mechanisms, credential types, messaging models, and protocols can be supported for each published interface and can be changed according to the associated service consumer needs.

[1] It is important to note that the service contract describes the application-level interface requirements. These will often be defined using a Web Service Definition Language (WSDL) description. However, many underlying components of the XML stack are not expressed in this form; some of these are likely to be addressed as derivatives of Web Services Policy (WS-Policy) are created, but many integration and interoperability challenges will remain for some time to come.

A virtual service and the supporting abstraction layer effectively create a container or policy enforcement point where policy can be used to select appropriate options that address the variability across service versions, supporting platform implementations, trust domains, and organizational structures.

The Promises of SOA

SOA defines a model in which business or technical services are offered to consumers in a distributed application system. The coarse granularity of this services approach, as opposed to an approach that defines objects and methods, creates an architecture in which logical business functions are reusable.

The business benefits of this architecture include the following:

- Increased business agility
- Wider set of connected customers and partners
- Better business alignment
- Improved customer satisfaction
- Reduced integration and operating costs

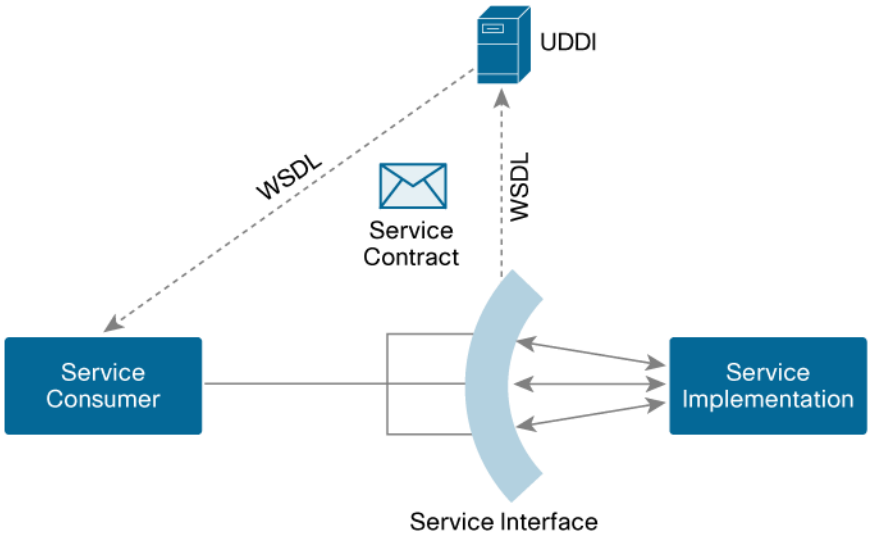
The technical benefits include the following:

- Efficient development
- Function reuse
- More effective maintenance
- Incremental adoption
- Controlled system enhancement

Loose Coupling

One of the more significant architectural attributes of a SOA is loose coupling. Loose coupling defines a model in which the interface to access a service and the supporting implementation are separated. This model is shown in Figure 1.

Figure 1. Service Publication and Discovery



Dependencies between a service implementation and a service consumer are minimized, as are dependencies on other parts of the system, helping ensure that service reuse capability is maximized. The service interface is maintained as a stable reference, and underlying implementation changes such as bug fixes are applied or alternative algorithms or processes are defined within the implementation.

Loose coupling is a very powerful concept if executed correctly. Much of the power of an SOA is derived from appropriately defined loosely coupled services. One measure of effectiveness for a SOA is not the number of services defined nor how quickly they are implemented, but rather how many times a particular service is reused. The high level of reuse of services leads to speedy development of new business systems, enhancing business agility.

The ultimate metric for many architects implementing loosely coupled services is the number of unexpected or unplanned reuses of a service. An effective, loosely coupled service, deployed at the right level of granularity, can be used in ways that the original designer had not imagined.[2]

Change Happens

- Loose coupling principals are extremely valuable at the level of an individual service. However, in the process of building a fully operational application system from Web services, many other types of change must be facilitated, including the following:
- Service versioning
- Service retirement
- Introduction of updated standards
- Selection of optional subsets within standards
- Increasing sophistication and governance requirements
- Varying service consumer needs
- Different geographical constraints
- Integration of older implementations

Service Versioning

Over time, any given service interface is likely to be enhanced or require changes. Current logic says that a new service definition or version can be declared, published to a repository such as Universal Description, Discovery, and Integration (UDDI), and then located dynamically by the service consumer, establishing a connection. However, for many deployments, the location or selection of a Web service occurs at development time, not run time. In this case, the service needs to recompiled, tested, checked for quality assurance (QA), and deployed to handle changes in the interface.

Increased service versioning also causes problems for the service provider. Multiple versions of an interface and the supporting implementations have to be supported. Often, the preferable approach is to support multiple virtual instances of a service that map back to a single deployed service interface and implementation. In this case, an abstraction layer would handle mapping, mediation, and transformation activities to accommodate the differences between versions. In addition, the abstraction layer would handle filtering on service addresses or message content and the routing of messages to the appropriate implemented services.

[2] Of course, there are security, governance, privacy, and many other concerns that must be resolved in real deployments that constrain loose coupling. However, a feature of virtual services and their policy-driven abstraction layers is that they provide an enforcement point where control can be maintained and monitored.

Service Retirement

As reuse of services becomes more common, the collection of consumers using a given service will grow. Particularly where the consumers are not directly under the administrative control of the service provider (for example, partners, customers, or users from other business units), the speed at which consumers migrate from early versions of a service that a provider wants to retire can be difficult to control, increasing the support burden on the service provider.

To handle this challenge, it must be possible to verify the collection of consumers that are using a service. Logging and monitoring capabilities in the abstraction layer should provide this information for administrative tracking as well as auditing and compliance purposes.

A phased retirement requires that users of older versions be routed to newer versions, with appropriate semantic and syntactic transformations applied as appropriate. Where automatic translation cannot be performed, or to handle requests to a service that has reached its end of life, the abstraction layer needs to return appropriate error responses to the service consumers.

New Versions of Standards

The state of Web service standards development is very fluid. In almost every area of standardization, new releases are being planned. Standards such as Security Assertion Markup Language (SAML) have been released in multiple versions that are not backward compatible. Often, capabilities and mechanisms provided by such standards are not expressed in a controllable way within the service contract established between the service and the consumer.

Even if all implementations of standards were completely interoperable (which they are not), the introduction of new versions would still lead to incompatibilities or the use of advanced capabilities that one side or the other in a service exchange may not be able to handle. An abstraction layer supports uncoupled deployment of new standards on each side of a Web service interaction without requiring that all participants upgrade in a synchronized manner.

Optional Subsets Within Standards

Even in a near-perfect world where the same versions of all standards are in use across all platforms, development environments and application standards typically allow selection of options, such as different authentication tokens or canonicalization types, message exchange patterns, and protocol bindings, among dozens of other preferences. Negotiation of optional subsets will be assisted over time by the WS-Policy framework, but many such choices need to be managed, and not enough support is currently available. In addition, what subsets are negotiated and what criteria are used should be a policy decision, not left to individual implementers. Polices will change over time, and the next collection of service consumers may require special handling.

The service abstraction layer provides a well-defined location where policies can be established or modified. It also provides a valuable location for monitoring the application of the policy for operational and auditing purposes.

Increased Sophistication and Governance Requirements

Early deployment of all new IT technologies (at least successful ones) such as Web services tends to follow a similar pattern: least-technology deployment. Organizations tend to use the smallest subset of the technology that they reasonably can to deploy an operational system.

For example, in the area of security, the simplest model for securing a Web service is to deploy existing session-based security mechanisms such as Secure Sockets Layer and Transport Layer Security (SSL/TLS) or VPNs. Pressure from other needs such as privacy or auditing requirements

is often necessary to cause organizations to move to more sophisticated mechanisms such as the message-level security features of the WS-Security standard. When business needs, partner requirements, governance requirements, regulatory controls, etc. cause a reevaluation of the simple early technologies deployed, the effect of changes can be costly if implemented services need to be recoded to support more sophisticated mechanisms and their operational needs.

A service abstraction layer allows new mechanisms to be deployed on behalf of the virtual service by plugging in the required mechanisms and integrating with supporting infrastructures. In most cases, this work can be handled without changing the implemented service.

Every Service Consumer Is Different

The business agility sought from Web services is often thought of in terms of rapid production of new business services through the use of other composable Web services. However, this view considers only the first part of the agility requirement. Getting the first business partner or client to actively use the service is the next step and often requires a significant amount of debugging support.

The next step is even more important: quickly implementing the service for the second, third, and succeeding new business clients. Service provisioning or implementing a service for consumers tends to be complicated by the fact that every consumer has a slightly different set of needs and capabilities. Some may have only simple identity access management (IAM) capabilities and be able to provide only username and password authentication mechanisms rather than the SAML credential type agreed upon with the first client. Others may require the use of Kerberos tickets or X.509 certificates, others may require that all transactions are signed or encrypted for integrity or privacy reasons, and still others may demand a higher level of service level agreement (SLA) and verification that those terms are being met.

In all these cases, the abstraction layer can handle the changes and differences that are demanded. For the first consumer, it can provide monitoring and debugging facilities to eliminate the bugs in the service contract and implementation. For the subsequent consumers, it can mediate among the requirements and mechanisms, preferably without affecting the implemented service.

Geographical Constraints

For enterprises operating across many geographies, compliance with regulatory requirements can be challenging. Specific regions place particular privacy and other regulatory requirements on the information that is transferred outside them; for instance, the European Union (EU) regulates information transferred outside EU countries.

An abstraction layer can provide the facilities to filter and then disguise, remove, or remap private or sensitive information as it is transferred across these geographic boundaries. This processing may be required on behalf of services offered within a geographic area to consumers outside it or on behalf of consumers with similar requirements.

Older Implementations

Web services and support of SOA have been evolving over the past 5 years. There are many implementations of XML over message queues (IBM WebSphere MQ, Java Message Service [JMS], etc.), as well as XML documents sent in raw HTTP payloads. Simple Object Access Protocol (SOAP) has moved progressively from the Remote Procedure Call (RPC) model to document-oriented mechanisms as the preferred messaging approach. Even more sophisticated

message exchange patterns are beginning to appear. The division between architects and implementers who prefer Representational State Transfer (REST, or RESTful design) rather than SOAP-based Web services continues, but a wider set of protocols such as Simple Mail Transfer Protocol (SMTP) (for store-and-forward or long-lived transaction models supporting Electronic Business using XML [ebXML]) and FTP (for transfer of large data sets or attachments) is being adopted.

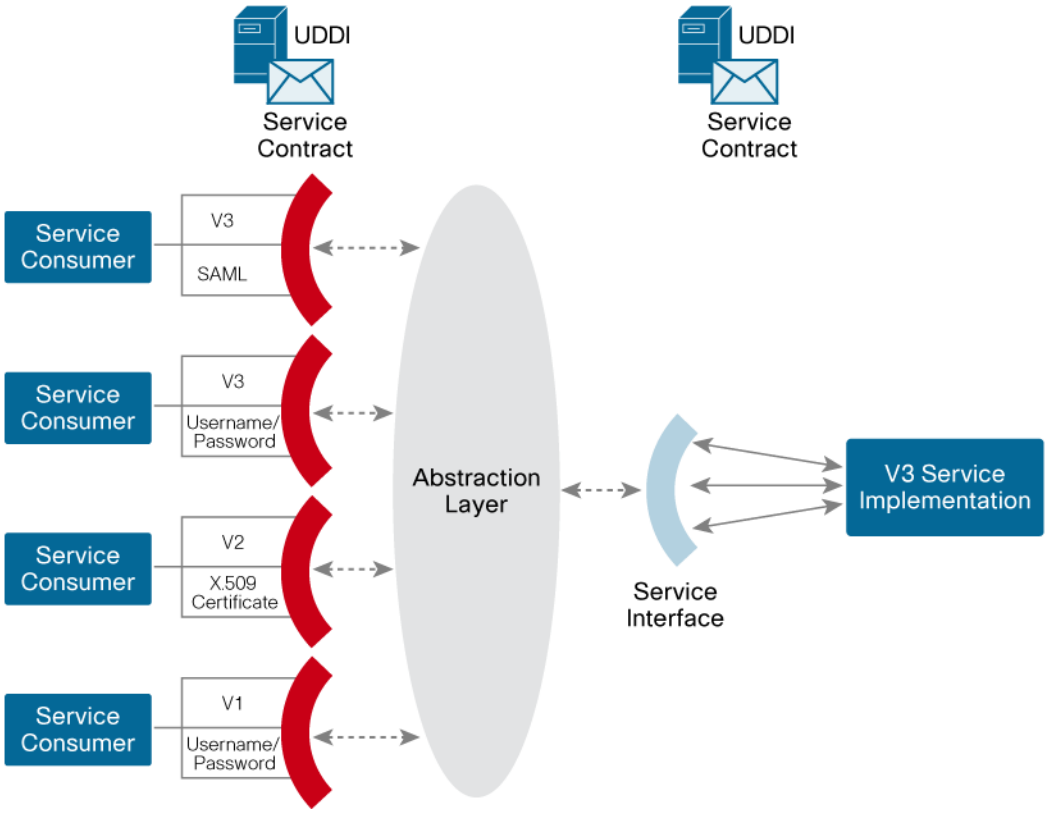
In, addition, the usefulness of Web services deployments often depends on effective integration with other infrastructures such as data repositories, enterprise service buses (ESBs), message queues, and identity and access management systems.

An abstraction layer can mediate among the various protocols, handling buffering for store-and-forward mechanisms and integration with other required supporting infrastructures.

Virtual Services and the Service Abstraction Layer

A virtual service is the exposed interface that a consumer connects to. Many different virtual services can be deployed to meet the requirements and changes among the collection of consumers that arise over time. The supporting abstraction layer provides the underlying policy-driven mechanisms that allow virtual services to effectively provide the generality required in an operational deployment. It also handles changes in underlying standards or mechanisms to reduce or eliminate any effect on the implemented back-end service. Figure 2 shows this interaction.

Figure 2. Service Abstraction and Versioning



Much that enables virtual services happens in the abstraction layer. Simply publishing additional interface descriptions is not sufficient; logic and transformation have to be applied to tie the virtual

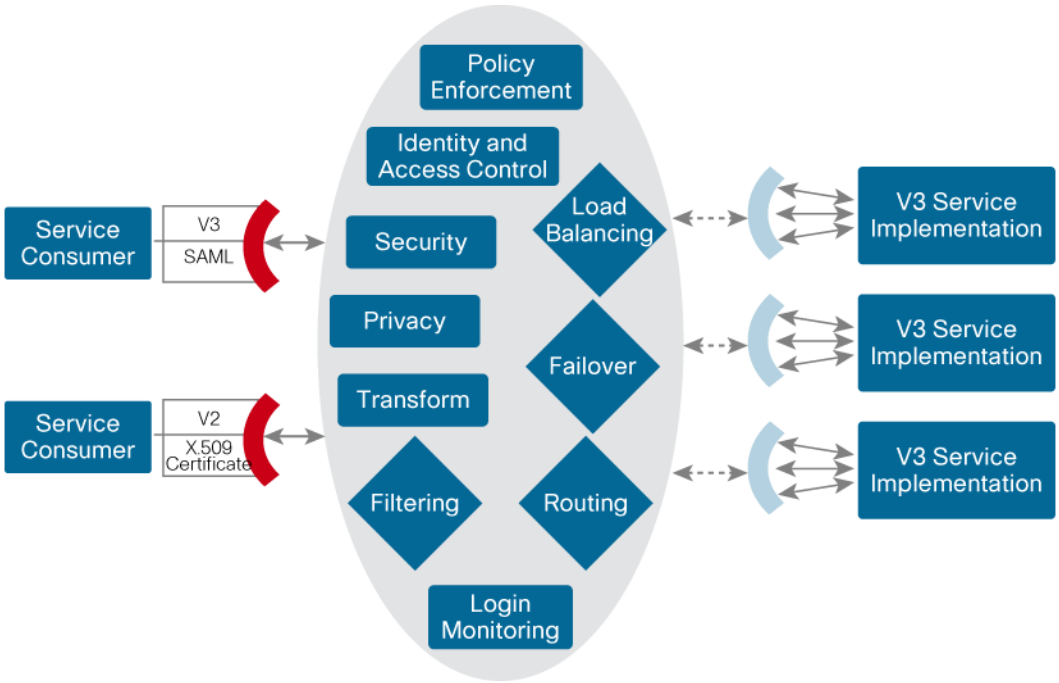
and actual services together. For this purpose, the abstraction layer needs to support a variety of capabilities and mechanisms.

The published virtual service interface that is used by a given service consumer will in most cases use a repository such as UDDI as its publication mechanism. Descriptions of many of the service requirements will be defined using WSDL. However, the use of underlying support capabilities such as the versions or subsets of particular standards in use cannot be fully captured at this time.

Supporting information must be specified describing the linkages, transformations, and requirements for deploying the virtual service interfaces. Examples of supporting information include the following:

- Addressing and location information
- Routings and filters
- Data hiding and privacy constraints
- Security mechanisms
- Mapping and translation descriptions
- Transport mediation processing
- Load-balancing and failover processing

Figure 3. Abstraction Layer and Functional Capabilities



- Addressing and location information: Each virtual service interface can be named and located through a number of mechanisms, although UDDI is the most common publication and discovery mechanism. However, the benefit of using the abstraction layer is that the name of the associated service can be changed, the address can be modified, or the number of instances of a particular service deployed can vary.
- Routings and filters: The description of how a virtual service is linked to its associated implemented service can take several forms. The simplest is a description of the name or address of the target service corresponding to the published virtual service. However, other

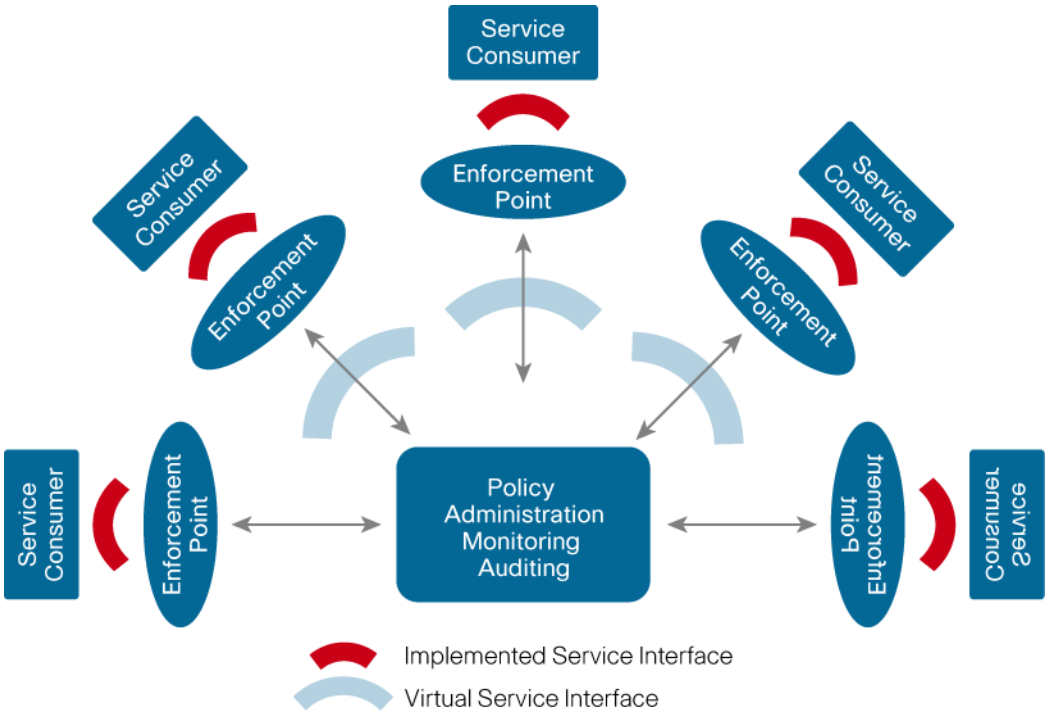
forms of routing information can be used that support additional linkage information, or filters describing message contents can be used to support a dynamic routing decision.

- Data hiding and privacy constraints: An implemented service can be deployed as a virtual service in different administrative domains. Information that can be validly exposed within an enterprise should not be allowed to transit enterprise boundaries. Message fields that should be kept hidden or private can be defined and removed or disguised in various ways for different service consumers.
- Security mechanisms: As a service is invoked by different consumers in different trust domains, different security mechanisms can be applied. For example, internal consumers of a Web service can be required to provide only a username and password as authentication credentials. External consumers of the service can have a choice of SAML or X.509 certificates as credentials. Internal use of a service may not require a signed or encrypted payload, but the same information moving across an enterprise boundary may require application of either or both mechanisms. The abstraction layer provides policy-based control over the type of authentication credential used and when each credential type is required, how each credential type is mapped, and what security transforms (for example, signing or encryption) are required for particular services or messages.
- Mapping and translation descriptions: To support differences in versions between the published virtual service and the implemented service, the messages passed between the virtual service interface and the implemented service interface may need to be transformed. This process may require removing, translating, mapping, or including fields within a message. This support may also require access to additional data repositories such as directories or databases.
- Transport mediation processing: For integration purposes, many types of transport may need to be tied together: for example, an older XML service that has been deployed over a message queue, or a new consumer who supports the use of XML only within HTTP messages and not SOAP. The abstraction layer handles all of the processes required to mediate between the various types of transports.
- Load-balancing and failover processing: In addition to routing, the abstraction layer can provide load-balancing capabilities to distribute the message load across the implemented services. In the event of a service failure, it can allow redirection to a standby version of the service running at some other location.

Virtual Service and Policy Control of Application Systems

The virtual service and the abstraction layer supporting it provide significant policy enforcement and instrumentation points for control and monitoring purposes. Figure 4 shows the linkage between many virtual services and the abstraction layer processing associated with them and centralized policy management, distribution, monitoring, and auditing facilities.

Figure 4. Policy Administration and Enforcement



Policies can be defined and enforced at the virtual service to handle transformation, security, privacy, integrity, and many other mechanisms. Any of the abstraction layer capabilities just described can be defined as a policy that is enforced consistently across application platforms.

As policy is being enforced and messages are being transformed in various ways, an ideal opportunity to instrument the behavior of the service is provided. Data collected about the operation of the virtual service and the messages passing through the abstraction layer allow support of SLAs, service testing, monitoring, auditing, and log record generation.

Adding SOA Components

To practically deliver results with SOA, the enterprise needs a scalable mechanism to support service virtualization and to provide the abstraction layer for security, interoperability, and visibility across the network of services. As the deployment of SOA and Web services progresses, scaling or complexity management requirements may require augmentation of the service abstraction layer with repositories for publication, discovery services for locating appropriate Web services, SLA definition and management tools, and other components.

In the context of virtual services, all these components continue to offer the same functions and value. However, from the perspective of the service consumer, the services that are supported, published, discovered, and so on are the virtual services.

Virtual service interfaces and the components derived from WSDL or WS-Policy can be described and published in a UDDI that is accessed by a service consumer. Service discovery will locate and identify the virtual services. SLA management will reference the service interface provided by the virtual interface defined for the service consumer. A workflow manager or business process language engine will operate on the virtual services defined.

If an organization decides that any of these supporting SOA or Web services infrastructure services is appropriate in deploying an SOA, the services will continue to operate unchanged;

however, all of the supporting abstraction-layer benefits, which provide a secure, robust, and flexible system, will enhance these components and the SOA as a whole.

Getting Started with Virtual Services

What do you need to consider when establishing your first virtual services in support of your enterprise SOA?

Virtual service concepts can be applied effectively with significant return regardless of the size or complexity of your SOA or Web service deployment. Because the concepts and mechanisms are designed to protect you against change and to expedite your deployments, they have value whether you have a single Web service with one or multiple consumers or hundreds of Web services in deployment.

For enterprise architects designing using SOA, virtual services and the abstraction layer provide known, well-defined locations for policy control and monitoring. For project architects and implementers, virtual services provide control points where changes can be defined and monitored without the need to redesign, code, perform QA, and redeploy services. For operations staff, virtual services provide a control boundary for applying changes under policy control to support new customers or service consumers. Virtual services also provide monitoring points for identifying problems that may arise or the need for additional capacity.

The Cisco® ACE Application Control Engine XML Gateway is designed explicitly to provide a virtual service model. The gateway provides a virtual service abstraction layer where message transformation, protocol translation, syntactic and semantic mediation, and security validation and processing occur. The extensibility, speed, and security of the gateway helps ensure that this abstraction layer increases the value of services connecting in the SOA. Service contracts and descriptions for implemented services in the form of WSDL can be imported from a UDDI directory or other source, enhanced to apply specific policy requirements, and then republished as a virtual service interface. The Cisco ACE XML Gateway acts as a policy enforcement and monitoring point to allow consistent policy application across all services and to monitor and audit application system behavior.

For enterprises beginning to deploy Web services, virtual services and the supporting abstraction layer allow fast deployment of early Web services to prove the concept and show a return on investment that supports the business case for continued investment and deployment of Web services and SOA.

For More Information

To learn more about the Cisco ACE XML Gateway and determine how it can provide scalable service virtualization to maximize the value of your XML Web services and SOA, visit http://www.cisco.com/go/ace.

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Asia Pacific Headquarters
Cisco Systems, Inc.
168 Robinson Road
#28-01 Capital Tower
Singapore 068912
www.cisco.com
Tel: +65 6317 7777
Fax: +65 6317 7799

Europe Headquarters
Cisco Systems International BV
Haarlerbergpark
Haarlerbergweg 13-19
1101 CH Amsterdam
The Netherlands
www-europe.cisco.com
Tel: +31 0 800 020 0791
Fax: +31 0 20 357 1100

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

Printed in USA C11-410408-00 6/07

