zapthink white paper

BUILDING SECURITY INTO A SERVICE-ORIENTED ARCHITECTURE

ZapThink, LLC • 11 Willow Street • Waltham, MA 02453 • info@zapthink.com • www.zapthink.com

BUILDING SECURITY INTO A SERVICE-ORIENTED ARCHITECTURE

May 2003

Analyst: Jason Bloomberg

Abstract

Service-oriented architectures (SOAs) based upon Web Services are an evolutionary improvement upon existing IT architectures, primarily because these SOAs offer loose coupling between Service producers and consumers. *Loose coupling* refers to a level of independence between the participants in a Web Services interaction that allows them to interact on their own terms, without requiring substantial changes on one system when the other changes. Such loose coupling enables an enterprise's IT infrastructure to be agile in the face of change.

Loose coupling, however, while simple to understand, is complex to implement. There are many requirements facing an enterprise's IT infrastructure that threaten the loose coupling of its architecture, and the most significant of these is security. Because fundamental security principles require a Service to authenticate a consumer, the Service and its consumer run the risk of being tightly coupled, unless the security itself can be handled in a loosely coupled fashion.

Such Service-oriented approaches to security require a management solution that both supports and enables the SOA as well as its security infrastructure. Such a management solution must enable and support loose coupling, so that the security infrastructure in an SOA can itself be Service-oriented.

Table of Contents

I. Loose coupling: a two-edged sword

Loose coupling is the key to the flexibility and agility promises of SOAs.

Web Services by themselves do not guarantee loose coupling.

Service-Oriented Architectures (SOAs) based on Web Services promise substantial benefits to the enterprise—simplified integration, more flexible IT infrastructures, and increased business agility. The reason such SOAs improve upon more established distributed computing approaches centers on several powerful ideas, or *idées fortes,* the most important of which is loose coupling. Two interacting systems are loosely coupled if the operation of each system is independent of the implementation of the other—in essence, the behavior of each system is fully specified by its interface. Loose coupling is the key to the flexibility and agility promises of SOAs; without it, new architectures will suffer the same limitations as older ones. Loose coupling results in large measure from the open standards that underlie Web Services and their inherently self-describing nature. Nevertheless, Web Services by themselves do not guarantee loose coupling. On the contrary: SOAs require several key infrastructure components to enable loose coupling and prevent the many ways that loose coupling can be compromised.

The value of loose coupling

One traditional way for distributed systems to interact is via *remote procedure calls* (RPCs). With an RPC, one computer actually executes a program on the other computer as if it was a local application. Traditionally, getting RPCs to work between different systems has never been easy. Even when the computer systems at either end of the network connection are communicating using the same protocol and language, the architect must carefully plan each side of the RPC so that they can locate each other, understand the communicated message formats, and resolve network and communication failures. Furthermore, when the two systems use different message formats, getting them to communicate is notoriously difficult—so difficult, in fact, that the entire data and application integration industry has evolved to solve this one problem.

The fundamental problem with such system-to-system communication architectures is that they are *tightly coupled*, which means that the architect must design each component system with the other systems in mind. As a result, making changes to one part of a tightly coupled system often affects the whole architecture, requiring expensive and difficult reworking. This problem of tight coupling was one of the primary drivers for the creation of Web Services. Web Services, on the other hand, allow for loose coupling between the systems that host the Web Service (known as producers) and the systems that access the

Service (known as consumers). Web Service consumers need have no knowledge beforehand about a Web Service, as long as the consumer can find it. As a result, a developer can make changes to an implementation of a Web Service without breaking the SOA, and the developer of a Web Service need have no particular knowledge about the consumers that will access it. Loose coupling, therefore, makes distributed computing far more flexible than tightly coupled approaches like RPCs.

Business agility—the ability of a company to respond efficiently to change and to leverage change for competitive advantage—is the fundamental business driver for SOAs.

Loose coupling's role in an SOA—to enable business agility

The greatest benefit of loose coupling is that it provides agility to IT infrastructures, because of the inherent independence of Service producers and consumers. Agile IT infrastructures, then, enable businesses that use them to themselves be agile. Such *business agility*—the ability of a company to respond efficiently to change and to leverage change for competitive advantage—is the fundamental business driver for SOAs.

Service-oriented architectures are an approach to designing distributed computing infrastructures that considers software resources as Services that are available and discoverable on a network.

Service-oriented architectures are an approach to designing distributed computing infrastructures that considers software resources as Services that are available and discoverable on a network. Service-oriented architectures are nothing new; the *Common Object Request Broker Architecture* (CORBA) and Microsoft's *Distributed Component Object Model* (DCOM) have long provided this functionality. These existing approaches to Service orientation, however, suffered from a few difficult problems. First, they were *tightly coupled*, which meant that both ends of each distributed computing link had to agree on the details of the API. Secondly, such Service-oriented architectures were proprietary. Microsoft unabashedly controlled DCOM, and while CORBA was ostensibly a standards-based effort, in practice, implementing a CORBA architecture typically necessitated the decision to work with a single vendor's implementation of the specification, because each vendor's interpretation of the standard varied enough to prevent seamless interoperability. Finally, CORBA and DCOM were *fine grained*, which means that service requests and responses typically contained small amounts of specific information, requiring many round trips between the consumer and the producer of the service.

Service-oriented architectures implemented with Web Services are a significant improvement upon DCOM and CORBA's weaknesses.

In spite of all of these issues, the concept of Service orientation continued to make sense, provided that the problems of proprietary approaches, tight coupling, and fine granularity can be solved. It is within this architectural context that Web Services were first imagined. Service-oriented architectures implemented with Web Services are a significant improvement upon DCOM and CORBA's weaknesses. The SOAs discussed in this paper are built upon the open standards of XML, SOAP, WSDL, and UDDI, offering unprecedented interoperability among different vendors' implementations. XML's inherently document-oriented structure enables systems to exchange coarse-grained messages between producers and consumers, and the standards-based discovery mechanisms provided by UDDI combined with WSDL's self-describing interfaces enable loose coupling.

SOAs provide a simplifying layer of abstraction

SOAs both reduce the complexity of underlying software functionality and enhance the power of the business to leverage the capabilities of that software, because SOAs provide a *layer of abstraction* that masks the complexity of the technology in an organization. By abstracting IT infrastructures, SOAs present functionality via loosely coupled Services that offer clear business value, independent of the underlying technology that supports them. Furthermore, SOAs provide consumers a way to dynamically discover and bind to available

Services while they are available, providing extraordinary flexibility to the businesses that invoke them.

However, many of today's existing IT architectures are n-tier—distributed infrastructures that separate data, business logic, and presentation logic onto different systems. It is critically important to understand that moving to an SOA does not require the replacement of existing n-tier architectures. Instead, SOAs introduce a layer of abstraction that hides the complexity of n-tier architectures. Understanding how this abstraction works, therefore, is critical for understanding SOAs. There are two elements to the abstraction in an SOA: *encapsulation* and *virtualization*.

Encapsulation is one of the fundamental principles of object-oriented programming, and is also a key concept in n-tier architectures. A software object is encapsulated when its inner workings are hidden from the outside world. All interactions with such an object take place through its interface. While encapsulation dates back to the object-oriented days, virtualization finds its roots in the very earliest software. The first programmable digital computers dealt in the world of zeroes and ones—that is, *only* zeroes and ones. Programs were zeroes and ones. Output consisted of zeroes and ones. As a result, programming was very difficult and programs were quite opaque. Into this black-and-white world came programs called compilers that let programmers work with English-like languages like COBOL. The compiler then took the COBOL code, crunched it, and spit out the zero-and-one object code that the computers actually understood. The COBOL compiler, therefore, *virtualized* the object code.

As computers grew more powerful and complex, virtualization and encapsulation techniques continued to provide additional levels of abstraction. Timesharing mainframe computers allowed users to have virtual control of the machines. Another example is the graphical user interface, which provided virtual access to underlying system resources. Component architectures also provided virtual representations of distributed computing infrastructures. At every step, software allowed people to work with relatively simple tools that accessed complex systems behind the scenes. That's the power of abstraction: enabling the tools people use to get simpler as they become more powerful.

The power of abstraction is enabling the tools people use to get simpler as they become more powerful.

Service orientation, then, is an evolutionary step in this inexorable progression to the next level of abstraction for distributed computing. By encapsulating software components, applications, and underlying systems with Web Services interfaces and then virtualizing these fine-grained functional Web Services into coarse-grained business Services, companies will have agile IT infrastructures that provide business agility. But make no mistake: this Service-oriented view of information technology is an abstraction—a virtualization of underlying software components and applications. SOAs are still software—nothing but ones and zeroes—at their core. Furthermore, since Services are nothing but software code at their core, existing architectures and programming languages remain just as important to the software's functionality. SOAs simply enable the functionality of this software to be exposed, consumed, and modified more easily.

SOAs are still software—nothing but ones and zeroes—at their core.

Loose coupling requires active management

It is important to remember that even when an organization has built an SOA, the systems and applications that provide that Service orientation are still present—they are simply hidden from view. IT departments must still manage these systems, only now they must also manage the Web Services they provide as well. Web Services management is therefore a combination of traditional system management techniques and the new Service-oriented management techniques that are a critical part of running SOAs in the enterprise.

In today's IT environment, management is tightly coupled to the systems being managed. System management products provide visibility and control into the various systems that make up an enterprise's IT infrastructure. In an SOA, however, what is important are the Services, not the systems *per se*. It is still just as important to manage the systems that underlie the Services, but even more important is the ability to manage how the systems enable business Services to function as they should—in a location independent, coarse-grained fashion. Service-oriented management solutions bridge the gap, therefore, between the underlying systems and the Services that run on top of them.

Active Service-oriented management is essential to preserving the loose coupling between Service producers and consumers, because such management abstracts the interfaces to underlying systems, handles the routing and prioritization of messages, and maintains the asynchrony of the Services, even when the requests are synchronous. Without these support capabilities, the Web Service consumer would need to have an understanding of how the Services operated, which is a clear example of tight coupling. Loose coupling, therefore, does not come automatically or easily; it requires sophisticated management capabilities to maintain it, especially in dynamic IT environments.

Security must be handled in a loosely coupled fashion as well

Traditional network security occurs on the packet level—firewalls determine whether packets should enter the network based upon network parameters such as the originating IP address. Furthermore, traditional application security naturally centers on particular applications, each of which usually has its own security administration mechanism. As enterprises move toward SOAs, however, both forms of IT security become insufficient. In addition to packet level network security, enterprises must also consider application level security that is aware of the contents of messages flowing between systems. Likewise, application-specific security administration must give way to enterprise-wide identity management and application security, as the Services exposed in an SOA can provide access to functionality on multiple systems.

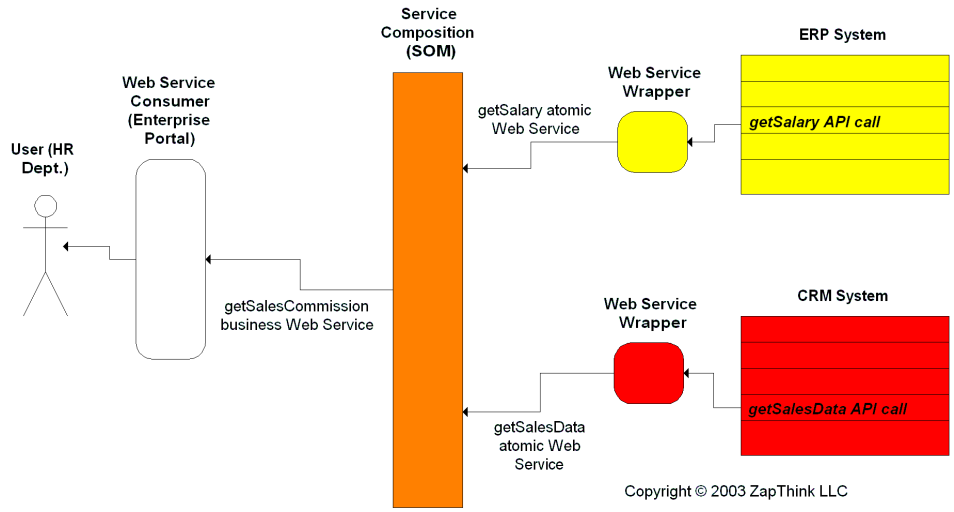
Application-specific security administration must give way to enterprise-wide identity management and application security, as the Services exposed in an SOA can provide access to functionality on multiple systems.

When implementing an SOA, there are many levels of change facing IT in the enterprise, and each type of change has security implications associated with it. In addition to the need for firewalls to be application and content aware, there are changes at the system level as closed, proprietary systems give way to open, loosely coupled systems. Closed systems are relatively straightforward to secure; an administrators only needs to set up the users and their privileges, and the work is mostly complete.

Securing open, loosely coupled systems requires a much more sophisticated security approach, involving multiple administrators that support distributed users. Different systems now have different policies and possibly different security mechanisms. As a result, administrators must manage security much more actively than was necessary in the closed model. In addition, enterprises should centralize their administration capabilities, and implement a hierarchical, delegated administration model to maintain a coherent, yet scalable enterprise security policy. Figure 1 below shows an example of an enterprise portal that accesses a getSalesCommission Web Service. This business Service touches upon two different back-end systems—the ERP system that provides a getSalary API, and a CRM system that exposes a getSalesData API. The ERP and CRM systems each have their own security policies, with separately defined users, indicated by yellow and red. The HR user logs into the portal, thereby being authenticated at the interface. The security policy for the getSalesCommission

Service (indicated in orange), however, is related to, but different from the policies governing the underlying systems.

Figure 1: Securing an HR Service



The challenge in an example like the one above is that the enterprise needs a single identity management and security policy infrastructure that governs the access to the four interfaces in the example (the portal, the business Service, and the two atomic Services) in a way that allows for flexibility in the event the systems, Services, or policies change.

Traditional distributed computing security was modeled by islands of security, which describe systems and users on isolated networks or subnetworks. The network acted as an island, with its own perimeter security, but users within the network were considered to be trusted. This "trusted vs. untrusted" dichotomy breaks down in a Service-oriented model, because users can access Services located on systems across one or more enterprises. The concept of trusted groups no longer has the same meaning; instead, enterprises must institute policies that apply to their entire enterprise network (including participants invited from outside), and administer that security in a tiered, or hierarchical fashion with a centralized root administrator. Departments or other organizational groups may then have their own administrators, but those administrators may in turn be administered by a more senior admin at a higher level within the enterprise.

II. Implementing application security when building an SOA

In order to fully understand how security can be provided and managed in the Service-oriented enterprise, it is important to first understand the principles of application security. Security is essentially the practice of mitigating risks, and application security in particular seeks to mitigate the following risks:

- Eavesdropping – Information remains intact, but an unauthorized person compromises its privacy. For example, someone could learn a credit card number, record a sensitive conversation, or intercept classified information.

- Tampering – An unauthorized person changes or replaces information in transit to the recipient. For example, someone could alter an order for goods or change a person's résumé.
- Impersonation – Information goes to a person who poses as the intended recipient. Impersonation can take two forms:
  - Spoofing – A person can pretend to be someone else. For example, a person can pretend to have someone else's email address, or a computer can identify itself as a Web site it is not.
  - Misrepresentation -- A person or organization can misrepresent itself. For example, a site might pretend to be a bookstore when it is really just a site that takes credit-card payments but never sends any goods.

These potential security breaches lead to the following requirements for application level security. Application level security contains five basic requirements, expressed in terms of the messages sent between parties. Such messages include any kind of communication between the sender (party who wishes to access an application) and the recipient (the application itself). The five requirements for application level security are:

- *Authentication*, The recipient of the message must be able to confirm the identity of the sender of the message.
- *Authorization*. The sender of a message must be authorized to send the message.
- *Confidentiality*. The contents of messages must not be available to unauthorized parties.
- *Data integrity*. The recipient of a message must be able to guarantee that a message hasn't been tampered with in transit.
- *Nonrepudiation*. The sender and the recipient must be able to guarantee that the sender send and the recipient received the message, including the time the message was sent and the fact the recipient received only a single copy.

In an SOA, each of these five requirements presents additional challenges to the enterprise beyond the requirements for securing individual applications, because of the essential Service-oriented nature of such architectures.

In an SOA, each of these five requirements presents additional challenges to the enterprise beyond the requirements for securing individual applications, because of the essential Service-oriented nature of such architectures.

Authorization on the meta-level

Authorization determines whether a user is allowed to perform the functions it requests or access requested data. Authorization is particularly important because of the need for tiered security administration in Service-oriented environments, where security administrators delegate their administrative functions to other administrators. At the simplest level, systems handle authorization with access control lists (ACLs) that list which users are entitled to perform certain operations (e.g., read, write, delete) on particular resources. However, ACLs are generally insufficient to handle the real-world security policies required at many enterprises, because Web Services provide programmatic interfaces that are difficult to monitor for suspicious activity. Take, for example, an HR application that has a Web Service interface. A request for Mary's salary would raise immediate suspicion from a human HR representative, but access to an improperly protected SOAP interface to the HR system would be more difficult to detect.

The ability to provide and administer authorization across multiple systems is a difficult problem that a Web Services specification known as WS-Security *is intended to address.*

This situation is even more complex when multiple, heterogeneous systems are involved, either within an enterprise or across two or more companies. Every company will likely have its own security policies, in addition to its own authorization technology. Therefore, the ability to provide and administer authorization across multiple systems is a difficult problem that a Web Services specification known as *WS-Security* is intended to address. WS-Security specifies an abstraction layer on top of any company's particular application security technology (PKI, Kerberos, etc.) that allows such dissimilar infrastructures to participate in a common trust relationship.

Confidentiality, data integrity and non-repudiation in a loosely coupled environment

Confidentiality means that an unauthorized person cannot view or interfere with a communication between two parties. Trust infrastructures like the Public Key Infrastructure (PKI) and Kerberos use encryption to ensure that messages are kept confidential. PKI in particular can use encryption to protect the confidentiality of data both in transit and in storage. Virtual Private Networks (VPNs) and Secure Sockets Layer (SSL) can protect the confidentiality of messages between two endpoints, but neither secures the data in storage or across intermediaries, because both SSL and VPNs are point-to-point techniques. Therefore, an SSL-encrypted message, for example, would have to be unencrypted at an intermediary, which opens a security hole.

There must be a way for the intermediary to read the part of the message that tells it what to do, without compromising the confidential payload of the message.

The problem of intermediaries is especially important in the context of XML and Web Services, because the SOAP protocol is designed to support one or more intermediaries that can forward or reroute SOAP messages based upon information either in the SOAP header or the HTTP header. Therefore, there must be a way for the intermediary to read the part of the message that tells it what to do, without compromising the confidential payload of the message. However, technologies such as SSL prevent the effective functioning of these intermediaries.

Unlike confidentiality, data integrity comprises two requirements: first, the data received must be the same as the data sent. In other words, data integrity systems must be able to guarantee that a message did not change in transit, either by mistake or on purpose. The second requirement for data integrity is that at any time in the future, it is possible to prove whether different copies of the same document are in fact identical.

PKI, for example, uses a technique called a message digest that provides one-way hashing to ensure data integrity. Hashing is an algorithm that takes any message and calculates a much shorter string of characters, known as the message digest, in such a way that performing the same algorithm on the same message again always yields the same result. The subsequent chance of two different messages yielding the same message digest is astronomically small. If the message digest is the same when a message is sent and when it is received, and the digest itself was kept secure, then data integrity is guaranteed. Likewise, a message can be hashed at two different points in time to determine if it has been altered.

When secure messages are sent, the recipient often requires that the sender can't repudiate the message, or claim that the message wasn't sent at particular date and time. Likewise, a sender would like to guarantee that a given message was received. The most common way to provide non-repudiation is through the use of digital signatures. With digital signature technology, senders can both provide evidence that a document is valid while simultaneously logging the

message transactions into signed audit logs. Once an audit log has been signed it cannot be surreptitiously modified.

Preserving loose coupling while ensuring confidentiality, data integrity, and non-repudiation in a Service-oriented environment is particularly challenging, because of the constraints such requirements put on both the Web Service producers and consumers. Specifications like WS-Security and the *Security Assertion Markup Language* (SAML) are intended to address the issue of preserving loose coupling by providing standard ways for both ends of a secure Web Services message to participate in the various forms of application security. Simply complying with the standards, however, is not sufficient to guarantee loose coupling. Instead, enterprises that wish to handle application security in a Service-oriented manner must take advantage of an active, Service-oriented management infrastructure that supports and enables loosely coupled application security.

Simply complying with the standards is not sufficient to guarantee loose coupling.

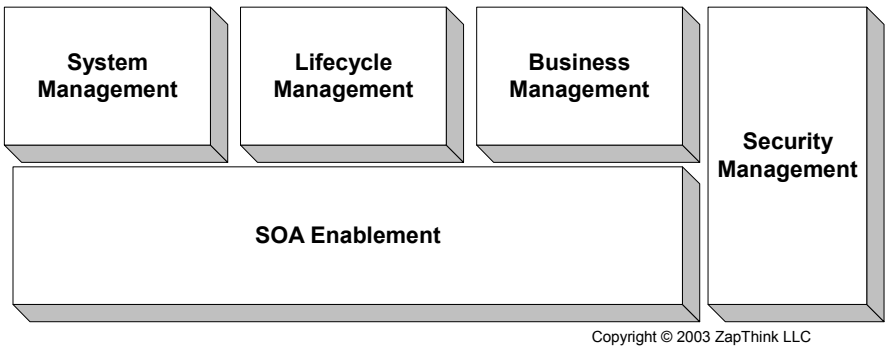
III. Service-Oriented Management: The Technology that Enables an SOA

A Service-oriented approach to IT management, in fact, goes beyond enabling loosely coupled application security; such management is critical to the ongoing operation of an SOA in general. To understand the management requirements for SOAs, it is important to understand the requirements for managing Web Services and the systems that underlie them, and then to understand the elements of a management solution that are necessary for operating an SOA.

The basics of Service-Oriented Management

There are a number of features that can be found in a Web Services management solution: system management, Web Services lifecycle management, business-oriented management, security management, and "SOA enablement," as shown in Figure 2 below. System management functionality includes the monitoring of Web Services and their underlying systems, including auditing, reporting, and alerting in the case of problems. System management also includes quality of service management, exception handling and root cause analysis of problems.

Figure 2: Categories of Service-Oriented Management Functionality



Lifecycle management addresses the issues of putting new Web Services into production, including provisioning, versioning, and deprecation of Web Services, as well as managing the configuration of individual Web Services and the

dependencies among multiple Services. Business management features include the management of business processes, providing business activity monitoring to executives, and enabling those executives to prioritize the quality of service that users of an SOA receive. As the previous section highlighted, security management supports the ability of a Web Services security infrastructure to provide loosely coupled authentication, access control, encryption, decryption and nonrepudiation.

The most important category of functionality might go by the name of SOA enablement: supporting the reliability and availability of the business Services the processes that link them. SOA enablement includes caching of Web Service functionality, converting synchronous to asynchronous requests, composing fine-grained APIs into coarse-grained business Services, dynamic routing of messages, and the translation of message protocols. Fundamentally, SOA enablement handles quality of service, scalability, and other enterprise-class performance requirements behind the scenes, in order to preserve the loose coupling of the Services exposed to the business.

SOA enablement handles quality of service, scalability, and other enterprise-class performance requirements behind the scenes, in order to preserve the loose coupling of the Services exposed to the business.

The enterprise must have a management infrastructure in place that can support the performance of the Services as they are being moved into production as well as once they are live. ZapThink calls the management infrastructure needed to support the ongoing functionality of a SOA *Service-Oriented Management* (SOM). Web Services management (WSM) is a more general term referring to applications that help companies manage the systems and applications that underlie their Web Services. SOM solutions, however, specifically support the development and execution of a Service-oriented architecture.

In an SOA, IT provides coarse-grained business Services in a location and connection-independent way.

In an SOA, IT provides coarse-grained business Services in a location and connection-independent way, as illustrated in Figure 1 above. In that example, SOM provides the Service composition functionality that takes the getSalary and getSalesData atomic Web services, and composes them (along with other Services, as necessary) into the getSalesCommission Web Service, which has business-oriented functionality. Many such Services will be for the use of more than one company, as appropriate, with the requisite Service-oriented security infrastructure controlling access to the Services. Web Service consumers can dynamically discover and bind to the necessary Services at runtime. The nuts and bolts of the software that makes such Services available takes place behind the scenes from the business user, because the applications and systems that actually provide the Service functionality are fully encapsulated and separated from the Web Service consumers by a composition, or virtualization layer. This encapsulation and composition is provided by Service-oriented management software.

SOM architectural approaches

The fundamental principles of IT management are visibility and control. In general, an IT management product should provide certain personnel with visibility into how the technology under management is behaving, and then allow those personnel to control the technology. There are basically two architectural approaches that SOM solutions can take to provide this visibility and control: a *proxy* approach or a *distributed* approach.

There are basically two architectural approaches that SOM solutions can take to provide this visibility and control: a proxy *approach or a* distributed *approach.*

An *XML proxy* is a hardware or software solution that actively listens for XML traffic on the network and either passes it along unmodified or performs some action on the XML content. XML proxies can operate either transparently or as auxiliary applications on the network. An SOM solution contains components that act as XML proxies when those components can be used to proxy Services within the IT organization. These components can proxy externally available Services on

the local network, or provide a single gateway for all internally available Services that the company wishes to expose externally. These proxies intercept SOAP messages in order to implement several key pieces of management functionality, including authentication, digital signing, encryption, reliability, compression, streaming, and state management.

SOM solutions that are based on XML proxies typically take a centralized approach to their architecture: running the core management engine on one server (or a cluster of servers). Centralized SOM architectures might have components that are distributed throughout the network or on other companies' networks, but the core software that provides the bulk of the management functionality is located in a particular place. The advantage of a proxy approach to SOM is the control the approach offers in terms of processing the XML on the network. The main drawback of such an intermediary-based solution, however, is that it adds latency to the XML messages—and sometimes, an enterprise cannot tolerate such latency on its network.

As a result, some SOM solutions have a decentralized, distributed approach to management. Instead of having a central control point, these vendors essentially set up a peer-to-peer arrangement of their software. Each of these distributed components provides a different part of the SOM functionality set. A distributed approach is infinitely scalable, and introduces no latency into the XML message stream, but at the cost of the high level of control that a proxy approach offers.

IV. Confluent Software: Service-Oriented Security

The Confluent CORE Web Services Monitoring and Management Platform enables enterprises to deploy SOAs that are secure, reliable, and managed.

Confluent Software's Service-Oriented Management solution, Confluent CORE, offers sophisticated Service-oriented security functionality in the context of a broad SOM solution. The Confluent CORE Web Services Monitoring and Management Platform enables enterprises to deploy SOAs that are secure, reliable, and centrally managed. CORE also helps increase operational visibility and responsiveness to changing business environments.

Confluent CORE supports the SOA during both design time and runtime for operational management including policy specification, distributed monitoring and metering, and policy-driven active management. Confluent CORE ensures that applications conform to enterprise-wide IT policies and best practices.

Centralized Management and WS-Security for interoperable authorization and authentication

Confluent's approach to security management centers on its centralized security policy management engine.

Confluent's approach to security management centers on its centralized security policy management engine. This engine offers centralized definition, distribution and evolution of authentication, authorization, and other security policies. In addition, it offers role-based access control, where administrators can define and enforce roles and access policies natively, or leverage existing definitions through LDAP support and integration of third party products, including Microsoft Active Directory, Netegrity, Oblix, and VeriSign's products. In addition, Confluent supports SAML for interoperability between different security systems, and WS-Security for non-repudiation and encryption to secure data over the wire. Confluent also supports HTTP(S) basic authentication and X509.3 based authentication.

Confidentiality, data integrity, and nonrepudiation

Confluent Software leverages its security policy management engine and its support of WS-Security to provide confidentiality, data integrity, and

nonrepudiation. When an XML message with a WS-Security header is received on the network, Confluent CORE first checks the validity of the header. Next, the system analyzes the type of credentials included in the message, for example, SAML or WS-Security tokens, and then identifies the appropriate Service for verifying the credentials, based upon the security policies in place. Only after these steps are performed will CORE pass along the message to the appropriate underlying security application.

Intermediary or peer-to-peer configuration

Administrators can place Confluent CORE in either a proxy or distributed, peer-to-peer configuration, mitigating the deficiencies of both approaches.

Administrators can place Confluent CORE in either a proxy or distributed, peer-to-peer configuration, mitigating the deficiencies of both approaches. The proxy intermediary approach is essential when enterprises wish to put CORE in the DMZ in order to explicitly intercept XML messages and perform actions based upon the content of those messages. However, in many cases, a distributed configuration is more appropriate, because of decreased latency and distributed configuration requirements that reduce administrative complexity and allow developers to increase their productivity by implementing elements of CORE on platforms of their choice.

Meta-level policy management

Confluent CORE introduces security and management policies as proxies the call "interceptors" at either the consuming or producing endpoints of Web Services. The product employs this approach by using interceptor frameworks that are natively supported by the deployment platforms, or via SOAP intermediaries that can run as separate applications. In either case, CORE allows for deploying such policies from a central location and abstracting the differences in deployment descriptors and interceptor frameworks. In addition, Confluent CORE supports a large library of policies that run on J2EE and .NET frameworks, as well as a policy manager that enables policies to be configured in a flexible manner.

SOA enablement

Confluent CORE supports multiple messaging and routing styles, including handlers for synchronous and asynchronous messaging. CORE supports multiple invocation models and transports, and can automatically handle dynamic routing between protocols, or based on quality of service metrics, including requester priorities or roles. As a result, CORE offers the composition, encapsulation, and dynamic routing capabilities that SOM solutions must offer to enable SOAs to operate at an enterprise-class level.

V. Conclusions

Service-oriented management and security are two fundamental requirements for effective implementation of an SOA.

Service-Oriented Architectures leverage the loose coupling capabilities of Web Services to provide an unprecedented level of agility to the enterprise. However, SOAs require an infrastructure that provides both management and security capabilities in order to maintain the loose coupling between Service consumers and producers. Service-oriented management and security, therefore, are two fundamental requirements for effective implementation of an SOA.

Both management and security capabilities tie the abstraction of an SOA to the underlying systems that support the architecture, and therefore it is essential that security and management solutions are themselves loosely coupled. It is the role of a Service-Oriented Management solution to provide these capabilities, and Confluent Software's CORE offering is an SOM solution that provides the

management and security functionality that is essential for building and running enterprise-class SOAs.

Copyright, Trademark Notice, and Statement of Opinion

About ZapThink, LLC

ZapThink is an IT market intelligence firm that provides trusted advice and critical insight into XML, Web Services, and Service Orientation. We provide our target audience 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's role is to help companies understand these IT products and services in the context of SOAs and the vision of Service Orientation. ZapThink provides market intelligence to IT vendors who offer XML and Web Services-based products to help them understand their competitive landscape and how to communicate their value proposition to their customers within the context of Service Orientation, and lay out their product roadmaps for the coming wave of Service Orientation. ZapThink also provides implementation intelligence to IT users who are seeking guidance and clarity into how to assemble the available products and services into a coherent roadmap to Service Orientation. Finally, ZapThink provides demand intelligence to IT vendors and service providers who must understand the needs of IT users as they follow the roadmap to Service Orientation.

ZapThink's senior analysts are widely regarded as the "go to analysts" for XML, Web Services, and SOAs by vendors, end-users, and the press. 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 October 2000 and is headquartered in Waltham, Massachusetts. Its customers include Global 1000 firms, public sector organizations around the world, and many emerging businesses. ZapThink Analysts have years of experience in IT as well as research and analysis. Its analysts have previously been with such firms as IDC and ChannelWave, and have sat on the working group committees for standards bodies such as RosettaNet, UDDI, CPExchange, ebXML, EIDX, and CompTIA.

Call, email, or visit the ZapThink Web site to learn more about how ZapThink can help you to better understand how XML and Web Services impact your business or organization.

ZAPTHINK CONTACT:
ZapThink, LLC
11 Willow Street, Suite 200
Waltham, MA 02453
Phone: +1 (781) 207 0203
Fax: +1 (786) 524 3186
info@zapthink.com

ZapThink, LLC • 11 Willow Street, Suite 200 • Waltham, MA 02453 • info@zapthink.com • www.zapthink.com