SOA Runtime Infrastructure

AUTHOR(S):

Anne Thomas Manes
(amanes@burtongroup.com)

Additional Input:
Chris Howard, Peter Lacey

Statement of Problem

What products and technologies should organizations use to implement a runtime infrastructure to support service oriented architecture (SOA)?

96020

Publishing Information

Burton Group is a research and consulting firm specializing in network and applications infrastructure technologies. Burton works to catalyze change and progress in the network computing industry through interaction with leading vendors and users. Publication headquarters, marketing, and sales offices are located at:

Burton Group
7090 Union Park Center, Suite 200
Midvale, Utah USA 84047-4169
Phone: +1.801.566.2880
Fax: +1.801.566.3611
Toll free in the USA: 800.824.9924
Internet: info@burtongroup.com; www.burtongroup.com

Table Of Contents

Statement of Problem

What products and technologies should organizations use to implement a runtime infrastructure to support service oriented architecture (SOA)?

Typical Requirements

SOA is a software design discipline in which application and infrastructure functionality is implemented as shared reusable services. After functionality has been implemented as services, an organization should be able to mix and match these services and rapidly create new applications to support changing business requirements.

SOA delivers a number of important business benefits, including faster time to market, lower costs, better consistency, and increased agility. Organizations view SOA as a way to better align information technology (IT) systems with business. Organizations also view SOA as a means to reduce duplicate efforts and to extract more value from their existing IT investments. For more information on SOA, see the following *Application Platform Strategies* documents:

- "Service-Oriented Architecture: Developing the Enterprise Roadmap" (overview)
- "VantagePoint 2005-2006 SOA Reality Check" (overview)
- "Enterprise Architects: Sowing the Seeds of SOA Success" (overview)
- "Building the Business Case for Service Oriented Architecture Investment" (Methodologies and Best Practices [MBP] document)

Organizations must recognize that SOA adoption cannot be accomplished simply through the deployment of a SOA runtime infrastructure. SOA is something an organization does—not something it buys or something it builds. SOA is an *approach* to developing IT systems. SOA requires fundamental changes to application design and development techniques, and it requires new levels of collaboration among project teams within the IT department and across lines of business. Given these required behavioral changes, a governance program is more critical to the success of a SOA initiative than a runtime infrastructure. Nonetheless, a SOA runtime infrastructure can facilitate SOA adoption plans, assuming the organization also addresses cultural and educational issues. For guidance on implementing a SOA governance infrastructure, see the Reference Architecture Technical Position, "SOA Governance Infrastructure."

A SOA runtime infrastructure typically must address the following requirements:

- Interoperability
- Quality of Service
- Loose Coupling
- Operations Management

Interoperability

A SOA initiative typically requires integration of disparate application environments. The most fundamental requirement of a SOA runtime infrastructure is that it enables interoperability across diverse technical platforms (operating systems, programming languages, application platforms, frameworks, middleware technologies, and so forth), and across logical and physical boundaries (departments, lines of business, corporations, security domains, and so forth).

Quality of Service

The ability to meet quality of service (QoS) requirements and service level agreements (SLAs) is a fundamental aspect of any enterprise architecture. A SOA runtime infrastructure must ensure that the environment delivers acceptable levels of service in terms ofsecurity,performance, andintegrity.

Security

On one side, a SOA initiative strives to open up IT systems to enable greater visibility and integration. But on the other side, it's critical to ensure that these systems are also properly protected. A SOA runtime infrastructure must provide mechanisms that protect services from unauthorized, fraudulent, or malicious use. It must protect the confidentiality and integrity of message traffic. It must protect IT systems from viruses, worms, Trojan horses, and other malware. The SOA runtime infrastructure also must integrate with existing security and identity management (IdM) systems.

In many cases, service interactions will cross security domains; therefore, the SOA runtime infrastructure must provide mechanisms that support identity credential mapping and cross-domain identity federation.

An effective security strategy is a critical aspect of a SOA runtime infrastructure. For more information about security strategies, security-related requirements, and position statements, see the following documents:

- *Application Platform Strategies* MBP document, "Developing a Web Services Security Strategy"
- *Application Platform Strategies* report, "Web Services Security: A Plethora of Products"
- *Identity and Privacy Strategies* Reference Architecture Technical Positions, "User Authentication," "User Authorization," "Identity Lifecycle Management," "Federated Identity," "Public Key Infrastructure," and " Roles"
- *Security and Risk Management Strategies* Reference Architecture Technical Positions, "System Placement," " Encryption," "Perimeters and Zones," and "Vulnerability Management."

Performance

If an organization plans to use SOA-based applications to implement mission-critical business processes, then the SOA runtime infrastructure must provide mechanisms that ensure adequate performance and scalability of SOA-based applications, as well as reliability, availability, and serviceability (RAS). The environment must be able to track and maintain specific SLAs for specific consumers and providers. For more information about SLAs, see the *Application Platform Strategies* topic folder, "Web Services Management."

Performance-enhancing mechanisms include techniques such as clustering, load balancing, partitioning, caching, message compression, and offloading compute-intensive processing to specialized hardware. For more information about performance-enhancing mechanisms, see the *Application Platform Strategies* report, "Scaling and Accelerating Web Services: The Roadmap to High Performance."

Integrity

A SOA runtime infrastructure must provide mechanisms that support reliable message delivery and transactional integrity across diverse distributed environments. For more information about integrity mechanisms, see the *Application Platform Strategies* overview, "Standardizing EAI with Web Services: An Inevitable but Prolonged Transition."

Loose Coupling

Loose coupling is one of the core SOA design principles. Loose coupling ensures flexibility, reusability, and resiliency in the face of dynamic systems. In its simplest definition, loose coupling means that changes to one system do not break interdependent systems. Loose coupling can be achieved through both design aspects and infrastructure aspects.

From a design perspective, this principle is enabled using four techniques:

- **Proper functional factoring:** In order to enable flexibility and reusability, application functionality must be factored into autonomous services with as little semantic dependency as possible on other services. Therefore, developers require SOA tooling that supports proper factoring techniques. See the Reference Architecture Technical Position, "Application Factoring," for more information on this topic.

- **Clean separation of consumer from provider:** In a SOA environment, the application consuming a service should not need to be aware of any technical implementation details about the application that provides the service. Likewise the provider should not need to be aware of any technical implementation details of the consumer. Consumer and provider should not need to share common objects or procedures, or even the same language or framework. Modifications to one should not require modifications to the other.
- **Clean separation of interface from implementation:** A service interface should hide service implementation details, such as programming language, operating system platform, and internal object model. The degree of abstraction between the service interface and the implementation directly affects how flexible (or brittle) application connections will be—the higher the level of abstraction, the more flexible the connection. In addition, the implementation should be able to expose its capabilities through multiple interfaces and protocols. A SOA runtime infrastructure must provide mechanisms that permit a high layer of abstraction between interfaces and implementations.
- **Clean separation of business logic from infrastructure functionality:** In a SOA environment, a particular service may be used in any number of applications, and each application may require unique infrastructure semantics. For example, a service might demand different security semantics depending on whether it is supporting internal or external consumers. In order to enable loose coupling, a SOA runtime infrastructure should provide mechanisms that permit applications to externalize infrastructure functionality, such as security, reliability, and transactions.

Proper design is a critical requirement for loosely coupled systems, but a number of runtime infrastructure features can facilitate and reinforce these design practices. From a runtime perspective, loose coupling is enabled using the following techniques:

- Managed Communications
- Flexible Communication Styles
- Mediated Interactions

Managed Communications

A SOA runtime infrastructure should provide mechanisms that facilitate a clean separation of business and infrastructure functionality. One mechanism is to use declarative policies and a managed communications system. In this type of environment, infrastructure functionality is associated with a particular service or service interaction using high-level declarative directives, which are codified as policies. These policies are then enforced at runtime by policy enforcement points (PEPs), which are typically deployed as agents at strategic points within the environment. For example, a PEP can be deployed as a proxy in front of a service endpoint to enforce security policies. For more information on managed communications and the SOA policy model, see the *Application Platform Strategies* overview, "VantagePoint 2005-2006 SOA Reality Check."

Flexible Communication Styles

SOA applications must have the flexibility to consume services in many different ways. A SOA runtime infrastructure should support any type of communication style or message exchange pattern (MEP), including synchronous, asynchronous, one-way messages, request/response, peer to peer, brokered delivery, hub and spoke, publish/subscribe, and orchestrated workflow. For more information about communication styles, see the following *Application Platform Strategies* documents:

- "Standardizing EAI with Web Services: An Inevitable but Prolonged Transition" (overview)
- "Crossing the Divide: The Mechanics of Process Execution" (report)
- "Enterprise Service Bus: EAI in Transition" (report)
- "Web Services Management" (Technology & Standards document)

Mediated Interactions

In some cases, it may not be convenient for a service consumer to interact directly with the service provider because of mismatches in communication styles and capabilities or because of the dynamic nature of the environment. A SOA runtime infrastructure should support mediated communications to enable:

- Dynamic location
- Dynamic binding
- Load balancing
- Message format and/or protocol transformations
- Version management and mapping
- Message routing
- Message queuing and caching
- Reliable message delivery
- Event processing
- Identity credential mapping

For more information about mediated interactions, see the following *Application Platform Strategies* documents:

- "Web Services Management" (Technology & Standards document)
- "Enterprise Service Bus: EAI in Transition" (report)

Operations Management

A SOA runtime infrastructure must provide mechanisms to manage service operations. Management functions include:

- Service Hosting
- Service Utilization
- Service Administration
- Service Monitoring
- Management Information Exchange
- Logging, Auditing, Metering, and Accounting
- Business Activity Monitoring

Service Hosting

A SOA runtime infrastructure must provide runtime containers for hosting services. Runtime containers take responsibility for managing the service agent lifecycle and allocating and de-allocating resources. (The service agent is the software that implements the service functionality.) See the Reference Architecture Templates, "Web Services Platform" and "Application Service," for more information about service containers and service agents.

Service Utilization

A SOA runtime infrastructure must provide mechanisms that manage utilization of a service (by service consumers) and enforce the runtime aspects of a utilization contract. A utilization contract represents the agreed terms of engagement between a consumer and a provider and comprises a set of agreements, such as:

- The interfaces, formats, and protocols that will be used
- The expected sequence of interactions, including minimum response times for asynchronous exchanges
- QoS and SLAs
- Security and privacy agreements

- Support agreements
- Upgrades and versioning agreements
- Remuneration terms

Some organizations may need to support dynamic discovery and utilization of services at runtime.

Service Administration

A SOA runtime infrastructure must provide mechanisms to configure services, mediation agents, and runtime policies, which must be enforced. It also must provide mechanisms to control (e.g., start, stop, reload, relocate, replicate) services and to reroute message traffic around problem nodes or paths.

Service Monitoring

In a typical organization, services are deployed across a disparate set of systems and involve complex dependencies. A SOA runtime infrastructure must provide mechanisms to monitor this diverse environment and aid in SLA compliance tracking, error detection, and root-cause analysis. For example, the infrastructure should report real-time metrics back to monitoring consoles, and it should alert performance management tools or administrators to excessive latencies and resource consumption, degradation of throughput, and other anomalous events and conditions.

Management Information Exchange

A SOA runtime infrastructure is likely to comprise multiple infrastructure components, which need to exchange information related to managing message interactions and enforcing utilization contracts. A SOA runtime infrastructure must provide a mechanism that enables diverse infrastructure components to share information about services, their constraints and capabilities, their operational performance, runtime routing and control policies, and any utilization contracts that must be enforced.

Logging, Auditing, Metering, and Accounting

A SOA runtime infrastructure must provide mechanisms to log service interactions, maintain audit trails, and measure service utilization for accounting purposes.

Business Activity Monitoring

The messages that flow across a SOA runtime infrastructure can provide tremendous insight into an organization's business health. A SOA runtime infrastructure should provide mechanisms for capturing and correlating message data for use in business activity monitoring (BAM) applications.

Alternatives

A SOA runtime infrastructure is, by definition, a heterogeneous environment. Even if a company has religiously adopted a single-vendor or superplatform strategy, its systems will still need to interoperate with systems in other organizations (partners, suppliers, customers, and so forth). Besides, no single vendor currently provides a complete solution that fully supports the requirements outlined in the "Typical Requirements" section of this Technical Position. A SOA runtime infrastructure must be assembled from a variety of products and services.

To compound the complexity of the situation, SOA infrastructure products often overlap in terms of purpose and capability. Before selecting products, an enterprise must first determine the functional capabilities that the infrastructure must address, and then select products that support those capabilities.

This "Alternatives" section starts by defining a conceptual model of a SOA infrastructure, and then drills into a functional model of a SOA runtime infrastructure. It then discusses alternatives for each functional component. This section is therefore organized into the following topics:

- Conceptual Model
- Functional Model
- Middleware
- Service Platform
- Service Mediation
- Service Administration
- Service Monitoring
- System of Record

Conceptual Model

The Root Template within the *Application Platform Strategies* Reference Architecture describes a conceptual model for a SOA infrastructure, called the network application platform (NAP). The NAP is also described in the *Application Platform Strategies* Root Document, "Turning the Network Into the Computer: The Emerging Network Application Platform." Figure 1 provides a graphical overview of the NAP.

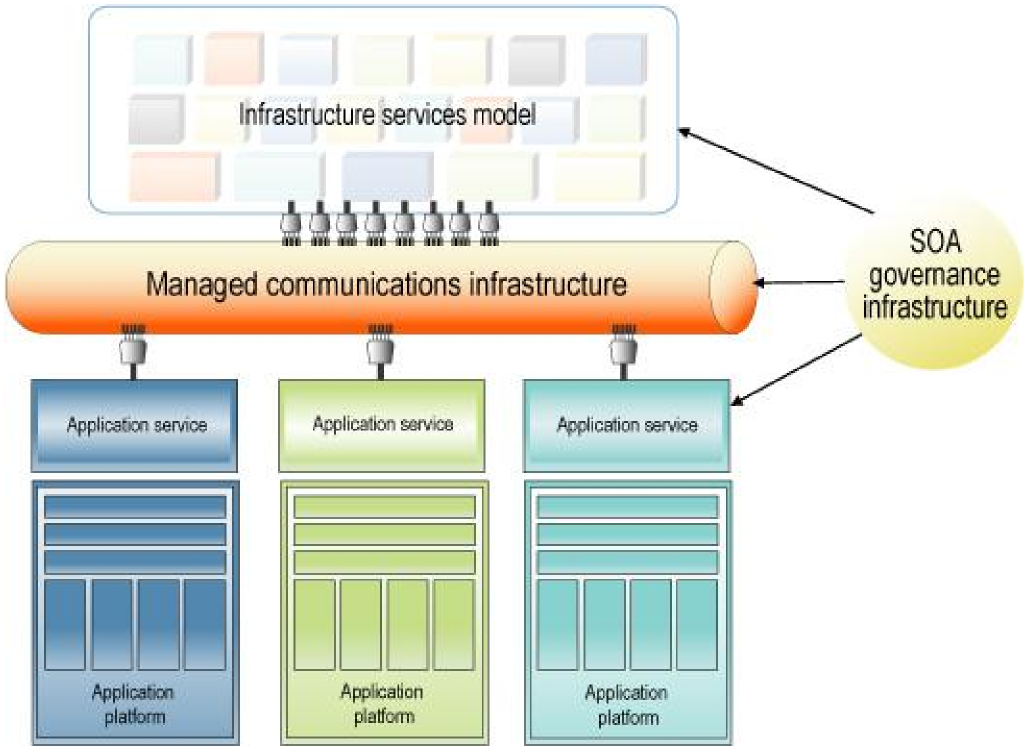

Figure 1: *Network Application Platform: A Conceptual Model of SOA Infrastructure*

The NAP is not a product, and no single vendor can provide it. The NAP is a conceptual model that describes the infrastructural capabilities required to support a SOA initiative. It is implemented using a wide assortment of products and technologies. In essence, the NAP defines a virtual application platform that supplies a foundation for SOA. It comprises three core concepts:

- Managed Communications Infrastructure
- Infrastructure Services Model
- SOA Governance Infrastructure

Managed Communications Infrastructure

The managed communications infrastructure (MCI) is the runtime backbone of the NAP. It is a product-, language-, and platform-neutral virtualized runtime infrastructure that enables and controls communications among service endpoints. The MCI supports any type of communication style, including one-way messages, request/response, peer-to-peer, brokered delivery, hub and spoke, and orchestrated workflow. It enables service endpoints to communicate in a managed way both within and across organizational boundaries. All communications are governed by policy, and the MCI automatically invokes the necessary infrastructure functionality required to enforce the policy. (The infrastructure services model supplies the infrastructure functionality.) The MCI is described in greater detail in the "Functional Model" section of this Technical Position.

Infrastructure Services Model

The infrastructure services model (ISM) is what makes the MCI more than just a basic communications pipe. It provides the various infrastructure services that enable the managed environment. The ISM supports theloose coupling principle that demands clean separation of business logic from infrastructure functionality. It represents a set of general-purpose infrastructure services that transparently deliver functionality—such as authentication, encryption, entitlements, auditing, validation, routing, transformations, persistence, transaction coordination, and other capabilities—to applications that communicate using the MCI. The specific infrastructure functionality required by a service interaction is defined using declarative policies and enforced automatically by the MCI. See the *Application Platform Strategies* Reference Architecture Template, "Infrastructure Services Model," for more information about the ISM.

SOA Governance Infrastructure

The SOA governance infrastructure (SGI) comprises tools and services that support a SOA governance program. The SGI provides mechanisms to manage and maintain service metadata and artifacts and to promote sharing of these artifacts. It also provides mechanisms to manage and maintain governance policies and to impose checkpoints during various phases of the software development lifecycle (SDLC) in order to verify that services and/or projects comply with these policies. And it provides mechanisms that support manual and automatic approval and exception processes. The SGI should integrate with traditional SDLC tools and IT management and governance systems. See the Reference Architecture Template, "SOA Governance Infrastructure," and the Reference Architecture Technical Position, "SOA Governance Infrastructure," for guidance on building an SGI.

Functional Model

The MCI represents the runtime infrastructure for the NAP. It supports managed communications between two service endpoints. Figure 2 shows a functional model of the MCI.

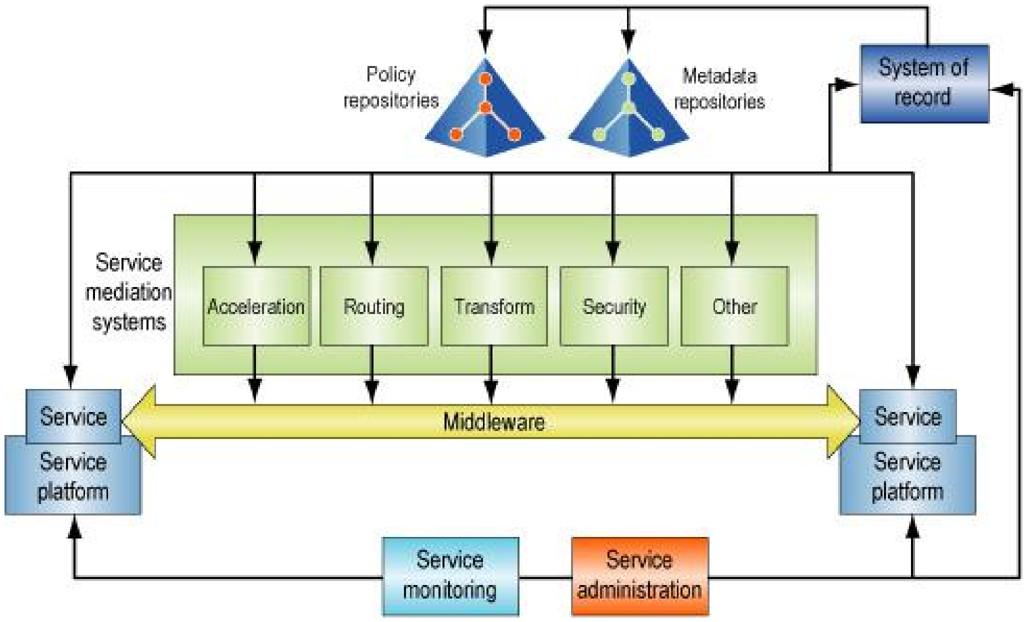

Figure 2: *Functional Model of a Managed Communications Infrastructure*

The key participants in a managed interaction include service endpoints that exchange messages and service intermediaries that manage the message exchange. The rules governing the message exchange are defined by policies, and it is the responsibility of the infrastructure to ensure that the policies are properly enforced. Hence service intermediaries act as PEPs. The intermediaries rely on infrastructure services (implemented according to ISM principles) to enforce the policies. Intermediaries can be deployed anywhere along the message path, including within the service endpoint.

The functional components in the MCI provide the infrastructure that enables this managed communications model. The functional components are:

- **Middleware:** Provides the means for service endpoints to communicate
- **Service Platform:** Hosts and manages the lifecycle of a service endpoint
- **Service Mediation:** Manages service intermediaries
- **Service Administration:** Configures and controls service endpoints, service intermediaries, and infrastructure components
- **Service Monitoring:** Monitors service endpoints, service intermediaries, infrastructure components, and message traffic
- **System of Record:** Maintains an index of service information (stored in metadata and policy repositories) and enables information exchange among infrastructure components

The following sections describe each functional component and identify the various alternatives that can be used to implement the functionality. Unfortunately, product offerings do not necessarily align directly with the functional components defined in the MCI; therefore, the same type of product may appear in multiple component categories. Note also that a single product category often does not address all the requirements of a particular functional component. Typically, multiple products must be used to fully implement a functional component (except for system of record).

Table 1 provides a matrix to assist readers in matching functions to products and vice versa. Capability is ranked from 0 to 3. A “0” indicates no support for the capability; a “1” indicates limited capability; a “2” indicates partial capability; and a “3” indicates full capability. An “*” indicates that some products in this category may provide this capability. Specific capabilities of each of the product types and how they map to each functional component are discussed in the sections that follow. The middleware component is not included in Table 1 because all components provide or support one or more middleware systems.

	Service platform	**Service mediation**	**Service admin**	**Service monitoring**	**System of record**
Application platform	3	1	1	0	*
Orchestration engine	3	1	1	0	*
Enterprise service bus	3	2	1	0	*
XML gateway	0	3	1-2	0	0
XML services management (XSM)	0	3	3	3	*
Enterprise systems management (ESM)	0	0	0	2	0

Registry	0	0	1	0	3

Table 1: *SOA InfrastructureProduct and Function Matrix*

Middleware

The middleware functional component provides the core communications capability for the MCI. It provides the means for service endpoints to communicate. The middleware component supports the following requirements:

- Interoperability
- Flexible Communication Styles

As a general rule, organizations should strive to reduce the number of middleware technologies used throughout the environment. Realistically, though, most organizations will use multiple middleware technologies. No one middleware solution supports complete interoperability, all communication styles, and all possible QoS levels. Individual applications may require specialized middleware to support their specific functional requirements. The MCI must accommodate multiple middleware systems.

In order to ensure better interoperability, services should be exposed using standards-based middleware. In particular, a standards-based middleware technology should provide the following core capabilities:

- **Standard protocol:** Uses standard data formats and communication mechanisms
- **Standard metadata:** Describes interfaces and data structures using standard description languages
- **Standard discovery:** Provides a standard mechanism to locate services and service metadata

In order to support managed communications, a middleware technology should enable messages to be intercepted such that they can be managed and mediated in compliance with the MCI model.

The inherent capabilities of a middleware system may be enhanced or augmented by other MCI components. For example, a mediation system may be able to intercept messages even if the middleware system does not natively support message interception.

Middleware alternatives include:

- Extensible Markup Language (XML) Messaging Systems
- Remote Procedure Call (RPC) and Distributed Object Middleware
- Message-Oriented Middleware

XML Messaging Systems

XML is an open, standard, universally supported markup language for defining text and representing structured data. Since its introduction in 1998, XML has become the most popular data format and integration language. A family of standards has grown up around XML to support additional features and functions. The core XML standards, all of which are managed by the World Wide Web Consortium (W3C), include:

- **XML:** A meta-markup language for defining document vocabularies and data formats
- **Namespaces in XML:** A mechanism for qualifying XML names in order to disambiguate entities with the same name
- **XML Schema:** A metadata language for defining message and document data structures
- **XML Path Language (XPath):** An expression language for addressing parts of an XML document
- **Extensible Stylesheet Language (XSL) Transformations (XSLT):** A language for transforming an XML document into another XML document using a stylesheet

- **XML Query Language (XQuery):** An expression language for querying documents and transforming documents using query expressions

For more information about the XML standards, see the *Application Platform Strategies* report, "Extensible Markup Language (XML) 2005: Core XML Standards and Their Impact on Business Development." For guidelines on using XML Schema, see the *Application Platform Strategies* MBP document, "Using XML Schema and Namespaces in XML."

A number of messaging middleware systems use XML for their data formats and XML Schema for their message metadata language. These XML messaging systems include:

- Web Services Framework
- Electronic Business using XML (ebXML)
- XML-RPC
- XML Syndication Protocols
- Plain Old XML

Web Services Framework

The web services framework (WSF) is an open, comprehensive, vendor-neutral, and language-independent middleware framework that enables integration across disparate application environments. Nearly all application platforms provide inherent support for the WSF, and many packaged application systems now provide built-in WSF-compliant interfaces. A huge ecosystem of products and services support the WSF.

The WSF provides a comprehensive middleware system that was designed to support some of the more esoteric requirements of a SOA runtime infrastructure, such as flexible communication styles, loose coupling, managed communications, and mediated messaging. It includes standard protocol, metadata, and discovery specifications, it natively supports message interception, and it supports clean separation of application and infrastructure information in its message structure. As a result, the WSF enables interoperability across heterogeneous systems and provides inherent support for the NAP model. For further discussion regarding the suitability of the WSF for SOA initiatives, see the *Application Platform Strategies* Root Document, "Turning the Network Into the Computer: The Emerging Network Application Platform."

The WSF is based on a combination of ratified and de facto standards collectively referred to as WS-*. The standards that make up the core of the framework include:

- **Simple Object Access Protocol (SOAP) 1.1:** A de facto standard XML messaging protocol
- **Web Services Description Language (WSDL) 1.1:** A de facto standard interface and protocol metadata language that uses XML Schema to define message formats
- **Universal Description, Discovery and Integration (UDDI) 3.0:** A standard discovery protocol and registry service specification (ratified by the Organization for the Advancement of Structured Information Standards [OASIS])

Another important WSF standard is the Web Services Interoperability Organization (WS-I) Basic Profile, which defines interoperability guidelines when using SOAP, WSDL, and UDDI.

The WSF also includes a host of composable extensions that add features such as security, reliability, and transactions. These extensions are in various states of readiness.

The foundational WSF security standard, Web Services Security (WS-Security), is ready for prime time. It has been ratified by OASIS, and WS-I has profiled its use in the WS-I Basic Security Profile. Most WSF products implement support for WS-Security. For more information about the WS-Security standard, see the *Security and Risk Management Strategies* overview, "Web Services Security Standards 2006: Where Are We Now?" For information about product support for WS-Security, see the *Application Platform Strategies* report, "Web Services Security: A Plethora of Products." For guidelines on using WS-Security in combination with other security measures, see the *Application Platform Strategies* MBP document, "Developing a Web Services Security Strategy."

A number of additional WSF extensions have recently been ratified, including:

- SOAP Message Transmission Optimization Mechanism (MTOM): Enables attachment and optimized processing of binary data in SOAP messages
- Web Services Addressing (WS-Addressing): Supports routing and complex message exchange patterns
- Web Services Reliable Messaging Protocol (WS-ReliableMessaging): Supports reliable message delivery
- Web Services Transactions (WS-Transaction): Supports atomic and compensating transactional integrity
- Web Services Trust Language (WS-Trust): Supports security token distribution and exchange
- Web Services Secure Conversation (WS-SecureConversation): Supports extended security sessions
- Web Services Business Process Execution Language (WS-BPEL): Defines an orchestration language

A number of vendors support either standard or pre-standard versions of these specifications, but for the near future, organizations should expect interoperability challenges when using these specifications. Until these new WSF extensions have been profiled, organizations should consider using an alternative middleware option, such as message oriented middleware, for applications with robust reliability and transaction requirements.

See the "Future Developments" section of this Technical Position for more information about forthcoming WSF extensions. For an in-depth discussion of the WSF core, see the *Application Platform Strategies* overview, "The Advent of the Network Platform: Web Services Move into the IT Fabric." For a description of the entire WSF, see the Reference Architecture Template, "Web Services Framework Standards." For an in-depth discussion of the WSF security model, see the *Security and Risk Management Strategies* overview, "Web Services Security Standards 2006: Where Are We Now?"

ebXML

ebXML is an open, vendor-neutral, and language-independent middleware framework designed to support business-to-business (B2B) integration. The framework isn't particularly suitable for internal integration projects. A small ecosystem of vendors and consulting organizations provide products and services for ebXML.

Although it is based on a similar foundation to the WSF (XML, XML Schema, and SOAP), ebXML relies on different higher-level services that impede interoperability between WSF and ebXML applications. Given that the WSF supports B2B integration, a specialized B2B framework is typically not necessary.

Nonetheless, some organizations may find ebXML useful for B2B interactions. A number of industry groups have mapped their B2B integration standards to ebXML, including the following:

- Open Applications Group, Inc. (OAGi)
- OpenTravel Alliance (OTA)
- RosettaNet (collaborative e-commerce)

The ebXML framework is based on a set of standards and specifications, most of which have been ratified by OASIS. The ebXML specifications include:

- **OASIS ebXML Message Service (ebMS) Specification 2.0:** An extended version of SOAP 1.1 that includes built-in extensions for security and reliability, which are not interoperable with the comparable WSF extension specifications

- **OASIS ebXML Registry Services (RS) and Protocols Specification 3.0:** A discovery protocol and registry and repository service specification that is different from UDDI
- **OASIS ebXML Registry Information Model (RIM) 3.0:** A repository information model that extends the RS specification to support metadata management
- **OASIS ebXML Collaboration Protocol Profile and Agreement (CPPA) Specification 2.0:** A metadata language for defining contracts
- **OASIS ebXML Business Process (ebBP) Specification Schema 2.0.4:** A metadata language for describing interactions between communicating partners (choreography)

On March 29, 2004, the International Organization for Standardization (ISO) approved ebMS 2.0, RS 2.0, RIM 2.0, and CPPA 2.0 as the ISO 15000 technical specifications. RS 3.0 and RIM 3.0 were ratified by OASIS in May 2005. ebBP was ratified by OASIS in December 2006.

XML-RPC

XML-RPC is an early offshoot from the original SOAP project. XML-RPC is an XML messaging protocol that communicates over Hypertext Transfer Protocol (HTTP) and supports a tightly coupled request/response MEP. (SOAP supports a much broader set of MEPs, especially when used in conjunction with WS-Addressing.) XML-RPC is a lightweight, easy-to-use middleware system that is popular among scripting language professionals. XML-RPC is a loosely typed system, which works well with scripting languages. It does not include a standard metadata language or a discovery protocol. It does not support extensions for security, reliability, transactions, or other advanced features.

Numerous open source implementations of XML-RPC exist for a wide assortment of languages, including Java, C++, JavaScript, PHP, Perl, Python, and Ruby. But XML-RPC hasn't attracted support from any major vendors, with the sole exception of Apple. XML-RPC can be useful for simple point-to-point integration applications. It has limited support from commercial software products, though. For more information about scripting languages, see the *Application Platform Strategies* reports, "The P-Languages: PHP, Perl, and Python for Enterprise Scripting" and "Ruby on Rails."

XML Syndication Protocols

XML syndication protocols, such as Atom and RSS (which today stands for "Really Simple Syndication," but in the past was known as "RDF [Resource Description Framework] Site Summary" or "Rich Site Summary") define XML protocols for syndicating, archiving, and editing "episodic" websites, such as weblogs (blogs). XML syndication protocols can be used for other data syndication applications, as well as alerting and notification requirements. For more information about XML syndication protocols, see the *Collaboration and Content Strategies* overview, "Transforming Communication Channels: RSS Within the Enterprise."

Plain Old XML

For applications with simple communication requirements, organizations may prefer to use "plain old XML" messaging, also known as POX. All other XML messaging systems define a formal message containment structure or envelope for conveying XML content. The envelope provides a means to encapsulate the message payload and potentially convey additional information that can be used by middleware to manage delivery of the message. POX does not use an envelope.

When using POX, applications typically exchange raw XML documents using standard transfer protocols, such as HTTP and File Transfer Protocol (FTP), or using proprietary protocols, such as message-oriented middleware. "Raw" means that the XML documents aren't wrapped in any kind of messaging protocol or envelope, such as SOAP, XML-RPC, or RSS. Instead, POX relies on the packaging capabilities of the transfer protocol to supply infrastructure information.

POX is minimalist middleware. The purpose of middleware is to externalize common, generic infrastructure functionality from business applications. POX provides significantly less infrastructure functionality than other XML messaging systems and therefore imposes additional burdens on business applications to perform infrastructure functionality. But it does support exceptional interoperability.

POX ≠ REST

Many people associate POX with the Representational State Transfer (REST) architecture. REST is an architectural style of system development, not a middleware technology. REST is similar to and compatible with SOA, although it imposes additional constraints on service interactions. The first and most fundamental tenet of REST is that applications interact with resources, and that all interactions with a resource involve the exchange of representations of the resource. One can think of REST as a resource oriented architecture (ROA) rather than a service oriented one. Another fundamental tenet of REST is the use of a uniform interface to interact with resources. REST is closely associated with HTTP because HTTP provides this type of uniform interface—all resource interactions use simple, generic methods: GET, POST, PUT, and DELETE.

Other important constraints of the REST style include clean separation of interface from implementation, stateless communications, and cache-ability of resource representations. As a result of these constraints, the REST style enables extreme scalability. The World Wide Web owes its extreme scalability to the REST architecture.

Because POX messages are frequently transferred using HTTP, many people assume that all POX messaging is RESTful. But in fact, many POX applications do not follow resource oriented design principles and do not adhere to the REST architectural constraints.

Note also that REST is not the exclusive domain of POX or HTTP. RESTful systems can be developed using any middleware, including the WSF. What matters is adherence to the REST design principles and architectural constraints. SOA does not impose these constraints on service design, but neither does it prohibit them. Therefore SOA systems can be RESTful. Applications that require extreme scalability will benefit from a RESTful design. For more information on REST, see the *Application Platform Strategies* overview, "The World Wide Web: An Introduction."

RPC and Distributed Object Middleware

RPC and distributed object middleware systems support tightly coupled client/server applications. RPC systems are procedure-oriented. Distributed objects systems are object-oriented.

Examples of RPC systems include:

- **ONC RPC**: The Open Network Computing (ONC) RPC, also known as the Sun RPC, is an Internet Engineering Task Force (IETF) standard.
- **DCE RPC**: The Distributed Computing Environment (DCE) RPC is a component of the Open Software Foundation (OSF) DCE standard, which is now managed by The Open Group.
- **Microsoft RPC**: The Microsoft RPC is a derivative of the DCE RPC and is the native RPC communication system for Windows 32 (Win32) applications.

Examples of distributed object systems include:

- **CORBA**: The Common Object Request Broker Architecture (CORBA) is an Object Management Group (OMG) standard.
- **RMI**: The Java Remote Method Invocation (RMI) is the standard remote invocation system for the Java platform.
- **DCOM**: Microsoft's Distributed Component Object Model (DCOM) is based on Microsoft RPC and is the native remote invocation system for Win32 applications.

These systems offer high performance, transaction integrity support, and an intuitive programming model. Each system defines a standard protocol (using binary formats rather than XML), a standard metadata format, and a standard discovery protocol. They do not support loose coupling, though. The client and server communicate by sharing an object or data model, and therefore the interactions are brittle—a change to one typically requires a change to the other. RPC and distributed object systems are also limited in terms of interoperability. Each system supports a limited set of operating systems and programming languages.

RPC and distributed object systems are a natural fit for point-to-point interactions in homogeneous, tightly coupled application architectures, but they can hinder a SOA initiative. For more information about RPC and distributed object systems, see the *Application Platform Strategies* overview, "Standardizing EAI with Web Services: An Inevitable but Prolonged Transition."

Message-Oriented Middleware

Message-oriented middleware (MOM) supports brokered, loosely coupled, asynchronous interactions among applications, but they do so using proprietary protocols. Examples of MOM products include IBM WebSphere MQ, TIBCO Rendezvous, and Microsoft Message Queue (MSMQ).

MOM systems offer high performance, transaction integrity support, and reliable message delivery. MOM systems typically support two types of communication mechanisms: message queuing and publish/subscribe. Because the communications are message-oriented and brokered, the connections are flexible.

The heart of a MOM system is a message bus, which provides the transport mechanism for sending and receiving messages. The message bus also provides a transactional queuing system for storing messages during asynchronous communications, and it manages topic subscriptions.

Applications communicate with the message bus either directly, using an application programming interface (API), or through an application adapter. Most MOM systems support the Java Message Service (JMS) API, although many systems also support proprietary APIs that offer richer features and functions. An application adapter provides a bridge between the message bus and a non-native API or protocol.

MOM systems play an important role in a SOA runtime infrastructure. MOM systems form the foundation of most traditional enterprise application integration (EAI) solutions, and their use is pervasive. MOM adapters enable the SOA runtime infrastructure to interoperate with legacy applications. But organizations should be careful not to rely too heavily on MOM. MOM systems do not meet the standardization requirements a SOA infrastructure needs from middleware. MOM systems communicate using proprietary protocols. Even though most MOM systems support the same API (JMS), different vendor products cannot interoperate except through protocol adapters. In addition, MOM systems typically don't provide standard discovery protocols, and they don't use standard metadata to describe message formats and interaction contracts. Developers must exchange interface and contract information out of band. MOM systems also don't natively support other important SOA requirements, such as managed communications and runtime enforcement of declarative policies. MOM systems typically support services such as security, transactions, and reliability, but they don't give developers the option to externalize the infrastructure functionality.

Building a SOA infrastructure based entirely on MOM would be challenging and expensive. The solutions pose interoperability hurdles and introduce symmetric integration stack requirements that will hinder SOA initiatives.

For more information about MOM, see the *Application Platform Strategies* overview, "Standardizing EAI with Web Services: An Inevitable but Prolonged Transition."

Service Platform

The service platform functional component provides tools, frameworks, and hosting containers for service endpoints in a SOA runtime infrastructure. It supports the requirements described in the "Service Hosting" section of this Technical Position.

Typically, service platforms are aligned with middleware systems. For example, if a service uses CORBA middleware, it must be developed and hosted using a CORBA service platform. Likewise, if a service uses MOM, it must be developed and hosted using a specific MOM platform.

The choice of platforms available for web services and other XML messaging systems is a bit more complicated. Therefore, this section will focus on the alternatives available for building and deploying XML-based services. Dozens of vendors and open source communities provide XML services platforms (XSPs). Most portals and application platforms include an XSP, as do many packaged application systems. In fact, many other middleware platforms also include an XSP.

Most organizations will use multiple XSPs. Each XSP supports a limited set of programming languages or deployment platforms. An organization must use one XSP for .NET applications, another XSP for Java applications, a third XSP for C++ applications, and a fourth for legacy application integration.

Each XSP is unique, with different capabilities in terms of performance, scalability, robustness, security, interoperability, extensibility, and productivity. XSPs can also be evaluated according to their portability across application servers, productivity of the included development tools, and intuitiveness of the management and administration interfaces. Many XSPs support only the WSF, although a growing number support additional forms of XML messaging, especially POX. Some XSPs support other types of middleware, such as MOM or CORBA. Some XSPs provide advanced features such as standards-based reliable messaging, discovery, orchestration, session management, and transaction support. Some XSPs provide client-only support, which means that service endpoints hosted by the XSP can initiate service requests, but they cannot respond to requests initiated by other service endpoints.

For more information about evaluation guidelines for XSPs, see the *Application Platform Strategies* overview, " Selecting a Java Web Services Platform: An Evaluation Framework."

XSP alternatives include:

- Application Platform
- Stand-Alone or Embeddable Platform
- Enterprise Service Bus
- Orchestration Engine

Application Platform

An application platform, such as .NET or a Java application server, provides tools, frameworks, and hosting containers for applications. Most application platforms include native support for the creation and deployment of web services. A growing number include support for POX services. Many application platforms also support CORBA, Java RMI, MOM, and other middleware systems, although application platforms typically don't enable seamless interoperability among the various middleware systems they support.

All five superplatform vendors (BEA Systems, IBM, Microsoft, Oracle, and SAP) include tools, frameworks, and hosting containers for XML services. Likewise, most mid-tier platform vendors, such as Adobe Systems, Borland Software, Red Hat JBoss, Sun Microsystems, and Sybase, also provide built-in XSPs in their products. In fact, the Java Platform, Enterprise Edition (Java EE) v5 specification requires that all Java EE-compatible application servers include support for web and POX services. Most portals also provide built-in XSPs. The OASIS Web Services for Remote Portlets (WSRP) specification defines standards for exposing and consuming portlets as web services.

Some application platforms include a service registry, which provides a system of record that supports information exchange. This capability is discussed further in the "System of Record" section of this Technical Position.

XSP capabilities are typically integrated with the rest of the application platform environment, sharing tools, management, and security systems. But this integration has a tradeoff. Services developed using these XSP tools often can be deployed only within the associated application platform containers.

For more information about application platforms and their XSP capabilities, see the following *Application Platform Strategies* documents:

- "The Java Superplatforms: Comparing IBM, Oracle, and BEA" (report)
- "The Microsoft Superplatform: Setting the Bar in the Superplatform Arms Race" (report)
- "SAP NetWeaver: Potential Wildcard in the Superplatforms Arms Race" (report)
- "The JBoss Enterprise Middleware Suite" (report)
- "JEE5: The Beginning of the End of Java EE" (overview)
- "Java Standards for Web Services: Bringing Synergy to a Fractured Environment" (overview)
- "Selecting a Java Web Services Platform: An Evaluation Framework" (overview)

Stand-Alone or Embeddable Platform

Some XSPs may be deployed as stand-alone platforms, or they may be embedded within other application platforms. These platforms typically support WSF and POX, but rarely support other middleware systems.

Organizations that are reluctant to upgrade their Java application platforms in a timely fashion can gain access to the latest XSP features by deploying an embeddable platform. Organizations can also use an embeddable platform to gain a consistent environment across multiple Java application servers. Open source embeddable Java XSPs include Apache Axis, Apache Axis2, Apache CXF, and Codehaus XFire. Commercial embeddable Java XSPs include HP Systinet Server for Java and Software AG Glue.

Many stand-alone platforms provide support for programming languages other than Java and .NET, such as C++, COBOL, PHP, Perl, Python, Ruby, or JavaScript. Lists of platforms and their supported languages can be found on the XMethods and SoapWare.org websites, although these lists have not been maintained since 2004.

Enterprise Service Bus

The enterprise service bus (ESB) product category represents the latest generation of the EAI software market. Unfortunately, the industry has not produced a clear definition of the product category, and this has caused a fair amount of confusion. The term has reached critical hype status, and now almost every middleware vendor claims to provide one. One can, in fact, take two vendors' ESB products, sit them side by side, and find little in common other than that they somehow sit in the middle of two or more network services.

Although the product category encompasses a number of products with very different characteristics, ESB products typically provide a simpler, more intuitive, and more open integration solution than earlier generations of EAI products, such as integration brokers. (Note, though, that many integration broker vendors now label their products "ESB.") An ESB typically supports WSF and POX, and sometimes ebXML, as well as CORBA, Java RMI, and MOM. Most ESBs also supply application adapters for integration with packaged applications and legacy systems. ESBs strive to enable seamless interoperability among the various middleware systems they support.

Quite a few vendors have been promoting the idea that an ESB provides a complete SOA runtime infrastructure or at least that it is an essential foundation for a SOA runtime infrastructure. They also imply that all SOA traffic must flow through the ESB. This is not the case. An ESB is just one piece of a SOA runtime infrastructure. It plays an important role, but it is not an essential component. An MCI does not require an ESB. An ESB's capabilities can be implemented using other technologies. More to the point, an ESB does not address all SOA runtime infrastructure requirements, and therefore cannot replace those other technologies—especially in terms of mediation services. Ironically, most organizations employ multiple ESBs within their infrastructure, and only a fraction of SOA traffic flows through a particular ESB.

ESB products typically perform a variety of functions and therefore are listed as alternatives for three types of SOA runtime infrastructure components:

- **Service platform:** An ESB provides tools, frameworks, and legacy application adapters that developers can use to encapsulate legacy applications and expose them as XML services. Many ESBs also provide tools and hosting containers for the development of new XML services. Many ESBs also include an orchestration engine, which is discussed further in the "Orchestration Engine" section of this Technical Position.
- **Service mediation system:** An ESB provides routing and transformation services. These capabilities are discussed further in the "Service Mediation" section of this Technical Position.
- **Service administration:** All SOA runtime infrastructure components, including ESBs, provide built-in administrative tools. These capabilities are discussed further in the "Service Administration" section of this Technical Position.

All superplatform vendors provide ESB offerings. (Note that these ESBs are separate and distinct from their application platforms.) Traditional integration broker and B2B integration vendors—such as Software AG, Sun/SeeBeyond, and TIBCO Software—also provide ESB offerings. Pure-play ESB vendors include Cape Clear Software, Fiorano Software, IONA Technologies, Progress Software, and Software AG. Numerous open source ESB projects are also available, including Apache ServiceMix, Apache Synapse, Codehaus Mule, JBoss ESB, ObjectWeb Celtix, Sun Open ESB, and WSO2 ESB. For more information on ESBs, see the *Application Platform Strategies* report, "Enterprise Service Bus: EAI in Transition."

Orchestration Engine

One of the goals of SOA is to enable organizations to mix and match services and rapidly create new applications in response to changing business imperatives. Although a SOA infrastructure can be used simply to bridge isolated application silos using point-to-point connections, the true value of the infrastructure emerges when organizations enable service reusability.

Service composition and orchestration systems enable the assembly and coordination of services into repeatable business processes. A business process defines an atomic interaction composed of multiple services. The composite business process can in turn be published as a higher-level service.

Service composition and orchestration systems typically provide modeling tools for designing the business process, development and configuration tools for defining the specific steps required to execute the process (including content mapping, transformations, routing, tracking, transaction management, and process invocation), and a runtime execution engine. Note that in order for service composition and orchestration systems to be of value, an organization must first develop a portfolio of reusable business services.

The leading service composition and orchestration standards are:

- **Modeling notation:** Business Process Modeling Notation (BPMN) is a graphical notation for specifying the steps in a business process. Originally developed by the Business Process Management Initiative (BPMI), this specification is now managed by OMG, and it was ratified in May 2006.
- **Execution language:** Business Process Execution Language (BPEL) is an XML syntax for specifying the steps required to execute a business process. Tools can transform a BPMN model into a BPEL execution script, although many BPMN notations cannot be expressed in BPEL. BPEL is also limited to orchestrating web services. It cannot orchestrate services implemented using other middleware alternatives. The WS-BPEL v2.0 specification was ratified by OASIS in April 2007. Many products support a pre-standard version of this specification, BPEL for Web Services (BPEL4WS) v1.1.

Given the limitations of BPEL, service composition and orchestration systems typically have a number of proprietary features and extensions that preclude portability and interoperability. Many orchestration systems support import and export of BPEL, but do not use it at runtime. Some orchestration systems use alternative modeling notations and/or execution languages. Most orchestration engines rely on a hub-and-spoke topology. Alternatives include:

- **ESBs:** Many ESBs (although not all) include an orchestration system. Example vendors include Cape Clear, Fiorano, Progress Software, and Software AG.
- **Integration brokers:** Example vendors include BEA, IBM, and SAP.

- **Business process management systems:** Example vendors include BEA, Cordys Software, Lombardi Software, Microsoft, Pegasystems, Savvion, Software AG, and TIBCO.
- **Pure-play orchestration systems:** Example vendors include IBM, IONA, Oracle, and SAP.
- **Portals:** Portal systems typically support service composition and orchestration, and some portals can expose orchestrated business processes as reusable services using WSRP.

When considering orchestration systems, several areas should be reviewed:

- Standards compliance
- Support for system and human workflow
- Integration with development and modeling tools
- Integration with registries, repositories, and lifecycle management systems
- Integration with service runtime infrastructure systems
- Integration with existing security and management systems
- Administration tools
- Reporting and analysis tools
- Presentation and access services
- Notification services

For more information on service composition and orchestration systems, see the following *Application Platform Strategies* documents:

- "Crossing the Divide: The Mechanics of Process Execution" (overview)
- "Integration Brokers: Providing the Middleware Fabric for Complex Integration" (report)

Service Mediation

The service mediation component ensures that messages are delivered properly, and it enables dissimilar systems to communicate. It supports the requirements described in the "Loose Coupling" and "Service Utilization" sections of this Technical Position.

Service mediation systems supply and coordinate intermediaries, which monitor, optimize, control, secure, route, and mediate message traffic according to a defined set of policies. Intermediaries act as PEPs. They intercept message traffic at strategic points along the message path and enforce policies. They can act as centralized PEPs (enforcing policies at a single point of entry, such as within a demilitarized zone [DMZ]), intermediary PEPs (enforcing policies in a proxy between endpoints), and endpoint PEPs (enforcing policies at a service endpoint).

An intermediary can do almost anything to ensure proper delivery of the message. Types of mediation include:

- Dynamic location and/or binding to a service
- Load balancing
- Message routing
- Message validation
- Message format and/or protocol transformations
- Version management and mapping
- Compression and decompression
- Message caching and/or queuing
- Reliable message delivery
- Event and subscription processing
- Mediating between endpoint communication styles and capabilities
- Logging
- Metering

- Authentication, authorization, and auditing
- Encryption and decryption
- Signature processing
- Credential mapping and federating across security domains
- Content scanning and threat detection
- Hardware-based accelerated processing

Most organizations will use multiple types of mediation systems. Some mediation systems specialize in security mediation, while others focus on reliability and integration. All mediation systems overlap in terms of capability, but they are also complementary. The most import thing to understand, though, is that no one mediation system supports all possible types of mediation.

Depending on the type of mediation system used, an intermediary may be deployed as a proxy, a filter, or an interceptor. In the proxy configuration, the intermediary operates as a separate entity, disconnected from any particular XSP. In the filter model, the intermediary is configured as a filter in an XSP's web server. In the interceptor model, the intermediary is configured to intercept messages within the XSP's message processing pipeline.

Service mediation systems include:

- Platform Interceptor
- Enterprise Service Bus
- XML Services Management
- XML Gateways

The following sections describe the capabilities of each of these alternatives. Table 2 provides a matrix to assist readers in matching mediation alternatives to mediation capabilities. The first group of capabilities (in blue) refers to the type of middleware systems that the product category can mediate. The second group (in green) refers to the deployment options supported. The third group (in pink) refers to the types of capabilities supported. Capabilities are grouped as follows:

- **Basic mediation:** Includes dynamic location and binding, load balancing, routing, validation, transformations, versioning, compression, logging, metering, authentication, authorization, auditing, and encryption
- **Reliability and events:** Includes message caching and queuing, reliable message delivery, and events and subscription processing
- **Mediated communication styles:** Includes mediation between endpoint capabilities, such as synchronous versus asynchronous or reliable versus non-reliable messaging
- **Security mediation:** Includes credential mapping and threat detection
- **Acceleration:** Includes hardware-assisted accelerated processing

An “x” indicates support for the capability. A blank indicates no support for the capability. “Maybe” indicates that the capability is product-specific.

Capability	Platform	ESB	XSM	Gateway
SOAP/POX	X	X	X	X
XML messaging		X	X	X
MOM/RPC		X	Maybe	Maybe
Endpoint PEP	X		X	
Intermediary PEP		X	X	X
Centralized PEP			X	X
Basic mediation	X	X	X	X
Reliability and events	Maybe	X	Maybe	
Mediated comm. styles		X		
Security mediation			X	X
Acceleration				X

Table 2: *Matrix ofMediation Alternatives and Mediation Capabilities*

Platform Interceptor

In addition to providing development and hosting capabilities as described in the "Service Platform" section of this Technical Position, an XSP provides inherent capabilities in its message processing pipeline to insert an interceptor (often referred to as a handler or module) to enforce policies. This is the mechanism that XSPs use to support advanced WS-* specifications, such as WS-Security or WS-ReliableMessaging. XSPs typically enable developers or administrators to configure standard interceptors (those that implement support for a WS-* specification) using code annotations or configuration files. A growing number of XSPs are adopting the Web Services Policy Framework (WS-Policy) as a standard means to represent interceptor configurations. Developers can also develop custom modules to implement additional capabilities.

Platform interceptors can process SOAP messages, and in some cases, POX messages. They can be deployed using the endpoint topology only. They support basic mediation capabilities, including validation, transformations, versioning, compression, logging, authentication, authorization, auditing, encryption, and signature processing. Some XSPs may support reliability and events.

Enterprise Service Bus

In addition to providing development and hosting capabilities as described in the "Service Platform" section of this Technical Position, an enterprise service bus (ESB) also provides a mediation solution that focuses primarily on integration. In particular, an ESB enables transparent interoperability between applications that communicate using different protocols and data formats. For example, an ESB permits a web service endpoint to communicate with a MOM application. ESBs typically supply point-and-click development tools for designing data transformation scripts (usually codified in XSLT) and protocol mappings.

ESBs can mediate XML messaging, MOM, and other middleware systems. They can be deployed using the endpoint topology when acting as a platform or as an intermediate proxy.

ESBs support basic mediation capabilities, including dynamic location and binding, load balancing, routing, validation, transformations, versioning, compression, logging, metering, authentication, authorization, auditing, and encryption. Most ESBs support reliable messaging and events, and some support two-phase commit transactional integrity. Most ESBs support and mediate between multiple communication styles, including request/response, one-way messages, synchronous and asynchronous, peer to peer, brokered delivery, hub and spoke, and publish/subscribe. ESBs typically do not support security mediation (credential mapping, domain federation, and threat detection), and they do not support hardware accelerated processing. For more information on ESBs, see the *Application Platform Strategies* report, "Enterprise Service Bus: EAI in Transition."

XML Services Management

XML services management (XSM) is a platform-independent administration, monitoring, and mediation system that supports basic and security mediation. XSM administration capabilities are discussed in the "Service Administration" section of this Technical Position, and XSM monitoring capabilities are discussed in the "Service Monitoring" section of this Technical Position. (XSM is also known as web services management [WSM], although its capabilities extend far beyond the WSF.) Vendors of XSMs include AmberPoint, BEA, Hewlett-Packard, Oracle, Progress Software, SOA Software, Software AG, TIBCO Software, and Vordel.

XSMs typically support mediation of all types of XML messaging traffic, including WSF, ebXML, XML-RPC, RSS, and POX. Some XSMs also support mediation of MOM and/or RPC traffic.

An XSM provides a set of intermediary agents that can be deployed throughout the infrastructure. XSM agents may be implemented as stand-alone software proxies, application server filters, or interceptors within an XSP. This configuration flexibility permits XSM agents to operate as centralized, intermediary, and endpoint PEPs. One advantage of using an XSM is that it provides a single mediation system with a single administrative console and consistent features and functions for the entire SOA runtime infrastructure.

XSM agents support basic mediation capabilities, including dynamic location and binding, load balancing, routing, validation, transformations, versioning, compression, logging, metering, authentication, authorization, auditing, and encryption. Some XSMs support reliable messaging and events. XSMs also support security mediation, including credential mapping and threat detection. All XSMs easily integrate with existing public key infrastructure (PKI), IdM, and directory systems.

For more information on XSM technology, market, products, and strategy, see the following *Application Platform Strategies* documents:

- "Web Services Management" (Technology & Standards document)
- "Web Services Management: Townsman of a Stiller Town" (Market Landscape document)
- "AmberPoint's SOA Management System and SOA Validation System" (Product Profile document)
- "Progress Software's Actional 6.0" (Product Profile document)
- "SOA Software's Service Manager 3.1" (Product Profile document)
- "Web Services Security: A Plethora of Products" (report)
- "Developing a Web Services Security Strategy" (MBP document)

XML Gateway

An XML gateway is a mediation system that focuses primarily on addressing SOA security and performance issues. (XML gateways are sometimes referred to as XML firewalls or XML accelerators.) Vendors of XML gateways include Cisco Systems, Forum Systems, IBM, Layer 7 Technologies, and Vordel.

XML gateways typically support all types of XML messaging traffic, including SOAP, ebXML, XML-RPC, RSS, and POX. Some XML gateways also support MOM and/or RPC traffic.

An XML gateway is typically packaged as a hardware appliance, although some vendors supply the technology in other hardware form factors or as software. The hardware configurations feature XML and cryptographic acceleration, and most hardware appliances have received Federal Information Processing Standards (FIPS) 140-2 certification.

An XML gateway supports centralized and intermediary PEP topologies. They specialize in managing traffic and enforcing security at a centralized point of entry, and most users deploy these gateways in the DMZ at the network perimeter. XML gateways can also be deployed at various intermediary points within the inner firewall, providing management, security, and acceleration of internal traffic.

XML gateways support basic mediation capabilities, including dynamic location and binding, load balancing, routing, validation, transformations, versioning, compression, logging, metering, authentication, authorization, auditing, and encryption. Some XML gateways support reliable messaging and events.

XML gateways specialize in SOA security features, including authentication, authorization, encryption, signature processing, credential mapping, provisioning, message scanning, message validation, denial-of-service detection, auditing, and other functions. These products also optionally include PKI management services. All gateways easily integrate with existing PKI, IdM, and directory systems.

For more information on XML gateways, see the following *Application Platform Strategies* documents:

- "Web Services Security: A Plethora of Products" (report)
- "Developing a Web Services Security Strategy" (MBP document)
- "Scaling and Accelerating Web Services: The Roadmap to High Performance" (report)

Service Administration

The service administration functional component supports the requirements described in the "Service Administration" section of this Technical Position. This component provides the capabilities to administer and configure services, service policies, service intermediaries, service resources, and various service runtime infrastructure components. This component is responsible for propagating codified contract policies to the appropriate mediation agents for runtime enforcement. It is also responsible for controlling (starting, stopping, quiescing, and moving) service instances, service resources, and runtime infrastructure components. The service administration component should support dynamic reconfiguration of systems, based on policies, in response to anomalous situations. For example, if a service exceeds a latency service level objective threshold, a new instance of the service agent should be deployed.

Service administration alternatives include:

- Built-In Administration Tools
- XML Services Management
- XML Gateways

Built-In Administration Tools

All SOA runtime infrastructure components provide built-in administration tools. For example:

- An XSP provides a console for configuring services and managing service agents deployed in the platform containers
- A service mediation system provides a console for configuring mediation policies and managing intermediaries
- A service registry provides a console for configuring the registry service

No standards exist for administering services and SOA infrastructure components; therefore, each product has unique administration tools, and most products support administration only of the specific infrastructure component (with the exception of XSM solutions and one XML gateway product).

XSM Services Management

In addition to the mediation capabilities that are discussed in the "Service Mediation" section of this Technical Position, XSM supports cross-platform administration capabilities. XSM also supports monitoring capabilities, as discussed in the "Service Monitoring" section of this Technical Position.

The XSM administrative capabilities complement the built-in administration tools supplied with other SOA infrastructure products. An XSM administration console provides a platform-independent environment for configuring service runtime policies, such as security and SLA requirements. One advantage of using XSM is that it can provide a single policy configuration tool for all XSPs and many mediation systems. Most XSM vendors have symbiotic relationships with XML gateway vendors, allowing administrators to configure both types of mediation systems from the XSM console. One XSM vendor, AmberPoint, has also started partnering with XSP and ESB vendors and can configure platform-based security policies via the XSM console. XSM can also automatically reconfigure intermediaries to deal with detected errors, SLA compliance issues, security attacks, and other runtime issues.

XSMs typically also support a symbiotic relationship with service registries via the UDDI protocols. (Registry alternatives are discussed in the "System of Record" section of this Technical Position.) A registry can notify an XSM whenever a new service is registered or an existing service is modified, allowing the XSM to automatically configure the service according to policies attached to the service in the registry or based on default management settings. At a minimum, the XSM can flag the service for custom administration and send an alert to an administrator. An XSM can also detect rogue services that have not been properly registered or configured, and send alerts to an administrator. The combination of XSM and service registries can streamline service provisioning and runtime management.

For more information about XSM alternatives, see the following *Application Platform Strategies* documents:

- "Web Services Management" (Technology & Standards document)
- " Web Services Management: Townsman of a Stiller Town" (Market Landscape document)
- "AmberPoint's SOA Management System and SOA Validation System" (Product Profile document)
- "Progress Software's Actional 6.0" (Product Profile document)
- "SOA Software's Service Manager 3.1" (Product Profile document)
- "Web Services Security: A Plethora of Products" (report)
- "Developing a Web Services Security Strategy" (MBP document)

XML Gateways

Two XML gateway vendors, Layer 7 Technologies and Vordel, provide an administrative console that supports cross-platform security policy management and configuration.

Service Monitoring

The service monitoring component supplies capabilities that support the following requirements:

- Service utilization
- Service monitoring
- Logging, auditing, metering, and accounting
- Business activity monitoring

This component supports monitoring of services, message traffic, service level objectives, key performance indicators, service resources and dependencies, and service runtime infrastructure components. This capability also provides visibility into the environment via dashboards, and it can propagate business activity information to business intelligence (BI) systems. It identifies anomalies and raises alerts before they become problems. It identifies the root cause of the problem, and it can automatically reconfigure the runtime environment (via the service administration component) to take corrective action.

Service monitoring alternatives include:

- XML Services Management
- Enterprise Systems Management

XML Services Management

In addition to the mediation capabilities discussed in the "Service Mediation" section of this Technical Position and the administration capabilities discussed in the "Service Administration" section, XSM also supports operational and business activity monitoring capabilities.

XSM agents monitor SOA traffic and collect information used for SLA compliance tracking, error detection, root-cause analysis, security monitoring, and BAM. These agents feed this information to XSM, ESM, and BI visibility dashboards. The XSM dashboards correlate the collected information and provide visibility into the health of service endpoints and composite SOA applications. Many XSM products also include BAM capabilities and business-oriented dashboards, which provide visibility into SOA business processes and key performance indicators.

Enterprise Systems Management

Traditional enterprise systems management (ESM) solutions, such as CA Unicenter, HP OpenView, and IBM Tivoli, enable operations visibility into the entire IT infrastructure, including application platforms, middleware, databases, operating systems, processors, storage, and networks. SOA infrastructure components rely on these IT infrastructure systems; therefore, any anomalies that occur in the IT infrastructure are likely to impact the SOA infrastructure.

Many ESM vendors now provide management products for monitoring services and SOA traffic. Examples include CA Web Services Distributed Management (WSDM), HP Business Availability Center, and IBM Tivoli Composite Application Manager (ITCAM) for SOA. By integrating SOA monitoring with traditional ESM, these products can dig deeper than XSM solutions to identify the root cause of a bottleneck or breakdown. (XSM products only analyze SOA traffic, not the underlying IT infrastructure.)

System of Record

The system of record functional component supports the requirements described in the "Management Information Exchange" section of this Technical Position. This component provides a central point of reference about services within a SOA environment and enables information exchange among service platforms, service mediation systems, service administration systems, and service monitoring systems. The system of record must support a standard data model and protocol to enable information exchange among heterogeneous runtime infrastructure components.

The system of record capability is typically implemented using a registry, which manages information such as:

- The location of all service endpoints for a service
- A list of all metadata associated with a service
- A list of all policies that apply to a service

The metadata and policies are maintained in repositories. In some cases, a repository may be integrated with a registry.

Registry users can classify services, service metadata, and service policies using predefined and custom taxonomies. These taxonomies can represent a wide variety of information, such as business function, service constraints and capabilities, remuneration requirements, SLAs, performance heuristics, and relationships between services. Users can compose complex queries based on these taxonomies.

Applications (service consumers) can use a registry at runtime to perform the following functions:

- Locate a suitable service
- Locate a specific service endpoint
- Retrieve service metadata and policies associated with a service endpoint and dynamically bind to the service

More often, though, service consumers rely on a mediation system to perform these dynamic functions. Mediation systems often cache location and routing information from the registry to reduce latency at runtime. XSM products typically work closely with registries to manage runtime service policies and to manage and control service agents. The symbiotic relationship between registries and XSM is discussed further in the "Service Administration" section of this Technical Position.

Service registries also play an important role in SOA governance, which is discussed in the Reference Architecture Technical Position, "SOA Governance Infrastructure."

For more information about the role of registries in a SOA runtime infrastructure, see the following *Application Platform Strategies* documents:

- "Registry Services: The Foundation for SOA Governance" (Technology & Standards document)
- "The Registry and SOA Governance Market Landscape" (Market Landscape document)
- "HP SOA Systinet" (Product Profile document)
- "IBM WebSphere Service Registry and Repository v6.0.1" (Product Profile document)
- "Microsoft Enterprise UDDI Services" (Product Profile document)
- "webMethods Infravio X-Registry" (Product Profile document)

System of record alternatives include:

- UDDI-Compliant Registry
- ebXML-Compliant Registry
- Platform-Specific Registry
- Informal Registry

UDDI-Compliant Registry

UDDI is the principle standard for service registries. It defines a standard data model and protocol for a stand-alone registry service (i.e., no integrated repository). It is supported by most SOA infrastructure products, including development tools, service platforms, service mediation systems, service administration systems, and service monitoring systems. UDDI provides an essential interoperability protocol that enables heterogeneous infrastructure components to work together and exchange information.

Two versions of the UDDI standard have been ratified by OASIS: UDDI v2 and UDDI v3. UDDI v3 is a superset of UDDI v2, and requires backward compatibility with UDDI v2. UDDI v3 includes a number of significant features related to security, governance, internationalization, and discovery services.

Vendors of UDDI v3-compliant registries include BEA, Hewlett-Packard, SOA Software, Software AG (CentraSite and Infravio), and TIBCO. IBM and Oracle include a UDDI v3 implementation with their application platforms. (BEA, Oracle, and TIBCO redistribute the Hewlett-Packard registry.) The SOA Software and Software AG products combine a UDDI registry with a SOA governance repository. Hewlett-Packard offers an integrated repository as an option. The TIBCO product combines the Systinet UDDI registry with a service administration solution based on AmberPoint XSM. SOA Software and Software AG offer an integrated XSM as an option.

Both Microsoft and Cape Clear provide UDDI v2 implementations with their platforms. Open source UDDI v2 implementations include Apache jUDDI and Novell Nsure UDDI Server. Novell has also launched a separate UDDI v3 open source project called Novell UDDI Server.

ebXML-Compliant Registry

ebXML Registry defines a standard data model and protocol for a combined registry and repository service. Very few SOA infrastructure products support the ebXML registry and repository standards; therefore, ebXML registries do not enable heterogeneous information exchange. An ebXML-compliant registry is more appropriate as a SOA governance repository than as a runtime registry.

Commercial ebXML registries include Sun Service Registry (included with Sun Java Enterprise System) and XENI GetMate Registry. Open source implementations include freebXML and Sino ebXML.

Platform-Specific Registry

A platform-specific registry is designed to support information exchange among a specific set of products provided by a superplatform. It does not support registry standards, such as UDDI and ebXML. Instead it implements a proprietary data model and protocol that is supported only by the products within the superplatform and perhaps a few specific partners. Currently, IBM and IONA supply platform-specific registries.

A platform-specific registry, such as IBM WebSphere Service Registry and Repository (WSRR), is appropriate only for organizations that implement a SOA runtime infrastructure using products from a single superplatform vendor. In the case of WSRR, it supports information exchange among WebSphere superplatform components, but it does not enable information exchange with third-party runtime components. (Note that IBM WSRR is separate and distinct from the UDDI v3-compliant registry that IBM distributes as part of WebSphere Application Server.)

Informal Registry

An informal registry is a mechanism, such as a wiki page or spreadsheet, that enables people within an organization to share service information. Although an informal registry can enable humans to share information, it cannot enable infrastructure components to share information. Therefore, this alternative is inappropriate for runtime infrastructure needs.

Future Developments

Over the next few years, SOA infrastructure decisions will be significantly influenced by developments in the following categories:

- WSF Standards
- Improved Tooling for POX
- Performance Improvements
- Grid Services
- Market Consolidation

WSF Standards

The web services framework (WSF) standards are in a state of flux. Two significant changes are imminent:

- Finalization of the WSF 2.0 standards
- Maturation of advanced WSF specifications

WSF 2.0

The core standards on which the WSF is based (SOAP 1.1 and WSDL 1.1) were never vetted and ratified by a formal standards body, and they contain a number of ambiguities, inconsistencies, and errors. The WS-I Basic Profile addresses the most grievous issues that impede interoperability, but nonetheless, revised specifications are required to address a number of shortcomings in the core framework, such as inadequate support for attachments, asynchronous messaging, routing, and versatile MEPs.

The Web Services Activity at the W3C is working on defining the next generation of the WSF. The WSF 2.0 core comprises the following specifications:

- SOAP 1.2**:** A simple upgrade of SOAP 1.1 that adds support for additional MEPs
- SOAP MTOM**:** A standard mechanism for conveying binary data with SOAP 1.2 messages, although MTOM also works with SOAP 1.1
- WS-Addressing 1.0**:** A standard mechanism to support asynchronous messaging and routing that works with both SOAP 1.1 and 1.2
- WSDL 2.0**:** A significant revision of WSDL 1.1

SOAP 1.2, MTOM, and WS-Addressing are ratified standards. WS-I has published a draft of the Basic Profile v1.2, which provides guidance for using SOAP 1.1, MTOM, WS-Addressing, and WSDL 1.1. The group has also established a charter to define the Basic Profile v2.0 to specify guidelines for using SOAP 1.2.

The current crop of web services infrastructure products (XSPs, ESBs, XSM, XML gateways, registries, and so forth) supports WSF 1.0 by default, although many products also support SOAP 1.2, MTOM, and WS-Addressing. Adoption of these new specifications is still pretty limited, but growing. In fact, Microsoft Windows Communication Foundation (WCF) supports SOAP 1.2 by default. Other vendors are certain to follow suit in 2007.

The adoption of WSDL 2.0 is less imminent. The WSDL 2.0 standard is still in development. The W3C Web Services Description Working Group published a Last Call Working Draft in March 2007 following a public review of an earlier Candidate Recommendation. The specification should progress to Proposed Recommendation in Q3 2007, and reach final Recommendation status in late 2007 or early 2008. Products are not likely to implement support for WSDL 2.0 until it reaches final status. Given that WSDL 2.0 is significantly different from WSDL 1.1, its adoption is likely to cause more confusion and disruption than SOAP 1.2.

The Advanced WSF

The WSF defines a composable architecture that supports numerous infrastructure extensions, supporting advanced functionality such as security, reliable message delivery, transaction coordination, publish/subscribe patterns, orchestration, and more. But the specifications that define these advanced capabilities are still emerging.

These specifications are being defined at OASIS, W3C, and the Distributed Management Task Force (DMTF). Efforts include:

- **OASIS Web Services Security (**WS-Security**):** Provides the means to attach security tokens to SOAP messages and to sign and encrypt portions of a SOAP message
- **OASIS Web Services Secure Exchange (**WS-SX**):** Defines a security token service (for the distribution of security tokens), a means to establish an extended secure conversation, and a means to define security policies
- **OASIS Web Services Federation (**WSFED**):** Defines protocols and mechanisms to enable federation across trust domains
- **OASIS Web Services Reliable Exchange (**WS-RX**):** Defines a protocol for reliable message exchange
- **OASIS Web Services Transaction (**WS-TX**):** Defines a protocol for coordinating transactions
- **OASIS Web Services Notification (**WS-Notification**):** Defines a protocol for supporting topic-based notifications and publish/subscribe patterns
- **OASIS Web Services Resource Framework (**WSRF**):** Defines a framework for modeling and accessing stateful resources using web services
- **OASIS Web Services Business Process Execution Language (**WS-BPEL**):** Defines an XML syntax for specifying the steps required to execute a business process
- **OASIS Web Services Distributed Management (**WSDM**):** Defines an architecture for managing distributed systems using web services
- **OASIS Web Services for Remote Portlets (**WSRP**):** Defines standard mechanisms to describe portable presentation information for web services
- **W3C Web Services Policy Framework (**WS-Policy**):** Defines a standard metadata format for expressing constraint and capabilities of a service
- **DMTF Web Services for Management (**WS-Management**):** Defines an architecture for managing distributed systems using web services

The WS-Security, WS-SX, WS-RX, WS-TX, WS-BPEL, WSDM, WS-Notification, WSRF, and WSRP specifications have been ratified, although only WS-Security is widely adopted and interoperable. WS-BPEL is also widely supported, but most vendors have implemented proprietary extensions, which subvert the benefits of standardization. Most portal systems support WSRP v1.0, but the minimal capabilities defined by the specification limit its effectiveness. WS-SX, WS-RX, and WS-TX were ratified in the first half of 2007 and have broad industry support. Most vendors are likely to implement support for these specifications by the end of 2007.

Although WSDM, WS-Notification, and WS-RF were ratified in 2006, they have not gained wide acceptance because they are the subject of contention between vendor factions. The competing specifications are WS-Management, Web Services Eventing (WS-Eventing), and Web Services Resource Transfer (WS-ResourceTransfer), respectively, and they are sponsored by Microsoft. Hewlett-Packard, IBM, Intel, and Microsoft have proposed a path to convergence for these specifications, although it will be at least three years before the contention is resolved.

WS-Policy is nearing completion and should be ratified in 2007. The WSFED standardization effort was launched in April 2007 and has not yet produced any deliverables.

See the Reference Architecture Template, "Web Services Framework Standards," for more information on the WSF specifications.

Improved Tooling for POX

POX is growing in popularity—perhaps as a backlash against the complexity of the WS-* specifications. In response, the vendors are improving the tooling in their application platforms and ESBs to make it easier for developers to implement POX interfaces. Note that vendors typically refer to this new capability as REST support, but the tools rarely, if ever, abide by REST principles and architectural constraints. XSMs and XML gateways already support management and mediation of POX messaging.

Performance Improvements

XML messaging affords tremendous flexibility, but it imposes a hefty cost in terms of performance. Companies that use XML must increase capacity in both networks and processing engines. Security requirements for digital signatures and message encryption compound these capacity costs. In short order though, the combination of Moore's Law and bandwidth growth will overcome this performance issue. And in the meantime, the industry is working on ways to reduce the performance overhead of XML and XML security.

For example, the XSP vendors continuously optimize the XML processors in their products. Each new product release is significantly faster than its previous generation. XML gateway vendors provide hardware appliances designed specifically to address performance issues related to XML. The W3C is also developing a new way to encode XML, called Efficient XML, which is designed to support limited bandwidth and constrained memory environments.

XML increases the flexibility and versatility of the environment; therefore, many users find the performance overhead to be a reasonable tradeoff. But developers should keep in mind that XML messaging may require higher capacity than traditional middleware in terms of networking, processing, memory, and storage. The *Application Platform Strategies* report, "Scaling and Accelerating Web Services: The Roadmap to High Performance," outlines recommended strategies for optimizing the performance and scalability of XML-based services.

Grid Services

To fully realize the flexibility and versatility of SOA, developers must be able to manage system complexity via the virtualization and sharing of resources. Additional architecture components are needed to support the collaboration of distributed application resources that span multiple tiers and organizational boundaries. The Open Grid Services Architecture (OGSA) defines an evolution of SOA toward a more virtualized environment.

The OGSA defines standards for creating and naming transient grid service instances. The specification virtualizes resource location behind service directories, factories, and reference handles. The grid service architecture provides a common infrastructure that supports policy-based control of resources, security, and dynamic component integration, in addition to supporting reliable invocation and delegation of service nodes. See the *Application Platform Strategies* report, "Grid Services: Pooling Distributed Resources for Virtual Supercomputing," for more information on OGSA and grid services.

Market Consolidation

The SOA runtime infrastructure market is volatile and experiencing many acquisitions:

- **Registry/repository:** The two leading independent registry vendors (Systinet and Infravio) were acquired—twice. Systinet was first acquired by Mercury Interactive, which was later acquired by Hewlett-Packard. Infravio was first acquired by webMethods and then was acquired by Software AG. BEA, Oracle, and TIBCO signed reseller or repackaging agreements with Systinet. And meanwhile IBM and IONA have developed platform-specific registries.
- **XSM:** Hewlett-Packard acquired Talking Blocks; CA acquired Adjoin; Oblix acquired Confluent, and was then acquired by Oracle; Progress acquired Actional; and SOA Software acquired Flamenco Networks and Blue Titan Software. Meanwhile BEA, Microsoft, SAP, and TIBCO have signed partnership, reseller, or repackaging agreements with AmberPoint.

- **XML gateway:** Intel acquired Sarvega (which then disappeared from the market); IBM acquired DataPower Technology; Cisco acquired Reactivity.

In addition, the lines between the various market segments are quite blurry, and as the vendors add more capabilities to their products, the lines are likely to disappear. In some cases, it's very difficult to classify a product in one particular market. Other than the fact that it does not support service monitoring, the Layer 7 XML gateway feels an awful lot like an XSM solution. Most XSM solutions and XML gateways could fully replace an ESB's mediation capabilities (although not its legacy platform integration capabilities).

The SOA runtime infrastructure market segments are likely to be redrawn over the next few years. As policy and management standards mature, the need for third-party administration, monitoring, and mediation agents may go away. It seems likely that the XML gateway, XSM, and ESB markets will morph or merge. See the *Application Platform Strategies* Market Landscape, "Web Services Management: Townsman of a Stiller Town," for further discussion of this topic.

Evaluation Criteria

All infrastructure decisions will be influenced by corporate principles, as described in the Reference Architecture Principles. Consider the management principles of “General vs. Optimal Solutions” and “Service Justification” in conjunction with the user principle of “Use of Services,” all of which are found in the Reference Architecture Principles. Does the organization wish to tailor the SOA infrastructure to meet the needs of a specific application? Will new integration projects be constrained to the existing architecture, or will infrastructure decisions be guided and influenced by the unique requirements of each deployed service? Will partners be required to format messages in a proprietary manner, or does the interface accept directives based on widely accepted standards?

When choosing SOA runtime infrastructure components, other management principles that should be considered include “Technology Maturity,” “Cost Center vs. Competitive Advantage,” and “Service Justification.” Because the SOA infrastructure is so pervasive, any of the components' drawbacks or benefits will be magnified. Organizations that are early adopters of new SOA infrastructure technologies can gain a competitive advantage by reducing operational costs, increasing development efficiency, enabling new revenue streams, and enhancing the customer experience. Those benefits need to be considered when creating a business case for the procurement of additional SOA infrastructure components.

Vendor principles of “Single Vendor vs. ‘Best of Breed',” “Closed vs. Open Solutions,” and “Vendor Risk” are particularly applicable to SOA infrastructure decisions. Organizations must be clear about their tolerance for implementing products from small startup suppliers versus well-established market veterans. Although many WSF standards are emerging, vendor-specific extensions may be introduced into shipping products, and organizations must be aware when this happens. Proprietary innovation by vendors should be balanced by any need for symmetric adoption by integration partners.

In addition to these issues, architects should also consider the following criteria when selecting SOA runtime infrastructure products:

Project requirements: What are the specific requirements of the current project? A SOA runtime infrastructure comprises many products, and not all products in the established SOA infrastructure must be used for all projects. Based on the project requirements, what are the appropriate pieces of the infrastructure that should be used to implement the project? What service platform is best for this project? What types of mediation services are required for this project? Is orchestration required for this project? How should this service be managed?

Superplatform alignment: Does the product being considered align with the organization's preferred superplatform solution (if it has one)? Does it integrate with the superplatform's tooling, administration, and security systems? Does it augment or replace existing components within the superplatform?

Standards compliance: Is the product being considered standards compliant? Does it support the correct standards? Does it support the entire standard or a subset? Does it provide proprietary extensions? If so, does it provide an option to easily turn off or disable the extensions?

Interoperability: Does the product support the WS-I interoperability profiles? Does it support interoperability when using more complex scenarios? Does the vendor participate in extensive interoperability testing with other vendor products? Has the vendor established partnerships with other vendors to ensure better integration between complementary products? In terms of legacy integration solutions, does the product support the project's interoperability requirements for languages, platforms, packaged applications, middleware, protocols, and/or APIs?

Ease of use: How easy is it to use the product? How easy is it to install and configure the product? Does it integrate with existing development and administrative tools? Does it automate tedious tasks? Does it provide an intuitive user interface? Does it provide convenient wizards and utilities? Are these wizards and utilities still helpful when trying to accomplish more sophisticated or complex tasks?

Security: Does the product integrate with existing security and IdM systems? Does the product support the security features necessary to address the application's surety requirements? Will messages need to cross security domain boundaries? What mechanisms are available to enable single sign-on and federated trust?

Management: Does the product integrate with existing management systems? What mechanisms are available for monitoring and enforcing SLAs? What metrics should be measured (latency, size, throughput, message signature, message content, faults)? Can service failures trigger notification alerts and generate log file entries? Is end-to-end QoS enforced? How are management policies configured? Are planning, analysis, and reporting tools available to aid in the management process?

Administration: Is the product easy to administer? Does the product integrate with existing administration systems? What is the level of integration with operating system, application server, and other application-platform infrastructure components? Does the product provide separate interfaces for administrative and development tasks? Can services be deployed, configured, and secured in an automated manner?

Performance and scalability: Can the product support expected performance and scalability workloads? What is the maximum acceptable response time? What is the maximum permitted latency that can be added to an interaction to support security, management, and mediation? What performance tuning features are supplied? Can performance and scalability be boosted using complementary products?

Cost of ownership: Are the licensing and support terms of the vendor products compatible with the organization's budget and business goals? How well are the products supported by the vendor and the community, and by systems integration firms?

For more information about evaluation criteria for specific product categories, see the following *Application Platform Strategies* documents:

- "Selecting a Java Web Services Platform: An Evaluation Framework" (overview)
- "Enterprise Service Bus: EAI in Transition" (report)
- "Web Services Management" (Technology & Standards document)
- "Registry Services: The Foundation for SOA Governance" (Technology & Standards document)

Statement & Basis for Position

Six position statements are necessary to address SOA runtime infrastructure:

- Middleware
  What types of middleware should be used in a SOA infrastructure?
- Service Platform
  What type of service platform should be used to implement and host a service?
- Service Mediation
  What types of mediation services should be used to manage, control, and secure message traffic?
- Service Registry
  What type of service registry should be used?
- Service Administration
  What types of administration systems should be used to configure services and service policies?
- Service Monitoring
  What types of monitoring systems should be used to provide visibility into the SOA environment?

Middleware Position

What types of middleware should be used in a SOA infrastructure?

The logic for choosing a middleware position is as follows:

```
IF the service will support content syndication
  THENexpose the service using an XML syndication protocol
OTHERWISE IF a service must have an extremely low barrier to adoption OR if the service will be consumed by the mass consumer market
  THENexpose the service using both POX and WSF
OTHERWISE IF a business partner requires use of ebXML or POX
  THENexpose those services using ebXML or POX
OTHERWISE IF an application requires reliable messaging semantics
  THEN IF the organization can dictate protocol requirements for both sides of the wire
        THENuse WS-RM or WSF over a MOM protocol
  OTHERWISEuse the WSF over HTTP and implement reliability semantics in the application code
OTHERWISE IF the functionality within a service needs to be distributed across multiple systems
  THENuse RPC middleware for distributed processing within a service
OTHERWISE IF legacy applications expose interfaces using RPC middleware or MOM
  THEN use an adapter to wrap the legacy middleware API with the WSF
OTHERWISE IF no WSF service platform exists for a particular application environment
  THENuse whatever middleware is available for the environment and wrap the middleware API with the WSF
```

OTHERWISEuse the WSF

Alternative Middleware position statements (important: choose *all* that apply):

Expose the service using an XML syndication protocol.

XML syndication protocols, such as Atom and RSS, are specifically designed to support easy syndication of "episodic" information. These protocols can also be used to support simple notification requirements. Atom and RSS are universally supported by vendors and the open source community and therefore provide the most convenient method for publishing information.

Organizations should adopt XML syndication protocols for these specific applications, but they should not attempt to use these protocols as the foundation for their SOA infrastructures for the reasons listed in the "use the WSF" position statement.

Or…

Expose the service using both POX and WSF.

POX provides the simplest and most interoperable middleware interface, and therefore provides the fewest barriers to consumers. POX messaging that conforms to the Representational State Transfer (REST) architecture also supports the highest scalability. But POX and REST support limited options in terms of security and reliability. Some consumers may prefer a more managed communications option; therefore, consumer-oriented services should also provide a WSF interface.

Or…

Expose those services using ebXML or POX.

Certain partners may not offer WSF interfaces for business-to-business (B2B) integration, and instead they might require use of ebXML or POX interfaces. Organizations will find it easier to integrate with POX than with ebXML; therefore, POX is generally preferred if the choice is available. ebXML may be popular with organizations that use Open Applications Group, Inc. (OAGi), OpenTravel Alliance (OTA), or RosettaNet B2B standards.

Organizations should adopt ebXML and/or POX for these specific B2B applications, but they should not attempt to use these protocols as the foundation for their SOA infrastructures for the reasons listed in the "use the WSF" position statement.

Or…

Use WS-RM or WSF over a MOM protocol.

The preferred SOA middleware alternative is SOAP over HTTP, but HTTP is not a reliable protocol. WS-RM, which supports reliable SOAP messaging over HTTP, has recently been ratified but only a handful of products currently support it. Support for WS-RM isn't likely to become pervasive until 2008. In the meantime, if organizations can dictate protocol requirements for both the service consumer and the service provider, then the organizations have two options available:

- **WS-RM:** A small number of XSPs implement early support for WS-RM—either the new standard or an earlier version of the specification. Many ESBs also support reliable mediation between XSPs that support WS-RM and those that don't. These vendors have done preliminary interoperability testing, but organizations should be prepared to do some tweaking if different products are used for the consumer and the provider.

- **WSF over MOM:** SOAP messages may be transferred using a reliable MOM protocol, such as IBM WebSphere MQ, Microsoft Message Queue (MSMQ), or Progress SonicMQ. This alternative requires that the same MOM protocol be available to both the consumer and the provider.

Or…

Use the WSF over HTTP and implement reliability semantics in the application code.

A MOM protocol is not appropriate in situations where the organization cannot dictate protocol requirements for both the service consumer and the service provider. If reliable message delivery is required in these circumstances, the only option is to implement the reliability semantics in the application code.

Or…

Use RPC middleware for distributed processing within a service.

RPC middleware systems, such as CORBA, Java RMI, DCOM, and Microsoft RPC, support more efficient communications than XML messaging, although the RPC model also produces tight coupling between components. It's appropriate to use RPC middleware for communications within a service, but these interfaces should not be exposed to the general SOA community for the reasons listed in the “use the WSF” position statement.

Or…

Use an adapter to wrap the legacy middleware API with the WSF.

Most organizations have used a variety of middleware technologies to create point-to-point connections between applications. By wrapping these existing interfaces using adapters and exposing them as web services, legacy application functionality can be made available to a much broader set of applications, and access can be controlled and coordinated by the MCI.

Or…

Use whatever middleware is available for the environment and wrap the middleware API with the WSF.

Although the WSF has nearly universal support from the vendor and open source communities, there are a few environments for which no WSF service platforms exist. In these situations, organizations must resort to using an alternative solution to create an interface to the application. After building the interface, though, the organization should wrap the middleware API with the WSF to make it available to a broader set of applications.

Or…

Use the WSF.

Although a SOA infrastructure could be implemented using any middleware technology, the WSF currently provides the best foundation to support SOA requirements, such as interoperability, security, flexible communication styles, and managed communications. The reasons to go with the WSF are threefold:

- The WSF is an XML messaging system, and XML is a universal standard that is easily processed by software and readable by humans.

- The WSF is the first middleware technology that has been universally adopted by all superplatform vendors, independent software vendors, and the open source community.
- The WSF natively supports the managed communications system described by the NAP conceptual model. Once a message has been placed on the network, it can be intercepted or caused to flow through any number of intermediaries before reaching its ultimate endpoint.

Other middleware technologies support only some of these features:

- ebXML is an XML messaging system, and it supports managed communications, but it has not been universally adopted by the vendor and open source communities.
- XML-RPC is an XML messaging system, but it has almost no vendor support, and it doesn't support managed communications.
- XML syndication protocols are XML messaging systems, and they have universal vendor and open source community support, but they do not support managed communications.
- POX is an XML messaging system, and it has universal vendor and open source community support, but it does not support managed communications.
- RPC and distributed object middleware support managed communications, but they don't use XML, and they don't have universal vendor or open source community support.
- MOM systems support managed communications, but they don't use XML, they are vendor-specific, and they don't interoperate.

Today, organizations should implement services that comply with the WS-I Basic Profile, which dictates use of SOAP 1.1 and WSDL 1.1, and the WS-I Basic Security Profile, which provides guidance for securing SOAP messaging using a combination of Secure Sockets Library (SSL), Transport Layer Security (TLS), and WS-Security.

Service Platform Position

What type of service platform should be used to implement and host a service?

Note: Selection of a service platform depends on the language and application platform used to create the service. For guidance on application platform selection, see the Reference Architecture Technical Position, "Application Platform Foundations."

The logic for choosing a service platform position is as follows:

```
IF the service being created will be a composition of other services
  THENuse an orchestration engine
OTHERWISE IF the service is being created using a .NET language
  THENuse .NET
OTHERWISE IF the service is being created using Java
  THEN IF the service will be deployed in a Java EE server AND the organization has standardized on one Java EE vendor AND the Java EE server has been upgraded to the latest release
      THENuse the XSP supplied by the Java EE vendor
  OTHERWISEuse a stand-alone and embeddable Java XSP
OTHERWISE IF the service will provide access to a legacy application
  THENuse an ESB that provides an adapter that works with the legacy application
OTHERWISEuse a stand-alone XSP that supports the language used to create the service
```

Alternative Service Platform position statements (important: choose *all* that apply):

Use an orchestration engine.

An orchestration engine permits developers (and, in some cases, business analysts) to encapsulate a repeatable multistep service interaction as a higher-level business service. Many ESBs include an orchestration engine, although in some cases the orchestration engine is sold separately.

Or…

Use .NET.

.NET provides integrated inherent support for the WSF. No other platform is appropriate to use with the .NET languages. Organizations should refrain from developing production services with WCF ("Indigo") until they are prepared to deploy .NET 3.0 across their servers.

Or…

Use the XSP supplied by the Java EE vendor.

Java EE 1.4 and 5.0-compliant application servers provide integrated, inherent support for the WSF. The Java EE 1.4 specification requires support for the Java API for XML-based RPC (JAX-RPC) as well as the WS-I Basic Profile. The Java EE 5.0 specification requires support for the Java API for XML Web Services (JAX-WS) and the WS-I Basic Profile. The latest versions of nearly all Java EE application servers also support WS-Security.

Or…

Use a stand-alone and embeddable Java XSP.

Stand-alone and embeddable Java XSPs—such as Apache Axis2, Apache CXF, Codehaus XFire, HP Systinet Server, and Software AG Glue—are appropriate in the following circumstances:

- **Organizations that plan to deploy Java services in a servlet engine or as stand-alone applications:** The stand-alone and embeddable Java XSPs provide support for Java web services without requiring a Java EE application server.
- **Organizations that run older versions of Java EE application servers:** The WSF is still evolving, and new XSP features and capabilities are added to each new application platform release. Unfortunately, the Java EE application server vendors rarely make the new capabilities available for previous releases. Organizations that are reluctant to upgrade their Java EE application servers in a timely fashion should use an embeddable platform in order to gain access to the latest WSF features.
- **Organizations that run multiple Java EE application servers:** The embeddable XSPs provide a common programming and administrative environment that is portable across multiple Java EE application servers.

Or…

Use an ESB that provides an adapter that works with the legacy application.

ESB products typically provide a set of adapters specifically designed to work with a variety of legacy application environments. Organizations should evaluate an ESB based on the adapters available.

Or…

Use a stand-alone XSP that supports the language used to create the service.

For languages other than Java and .NET, organizations should look for a stand-alone XSP that supports the language in question. Stand-alone XSPs are available for C, C++, COBOL, PHP, Perl, Python, Ruby, JavaScript, Delphi, Smalltalk, Ada, and many other languages.

Service Mediation Position

What types of mediation services should be used to manage, control, and secure message traffic?

The logic for choosing a service mediation position is as follows:

```
IN ALL CASESkeep the number of intermediaries to a minimum

-AND-

IF XML processing acceleration is required for functions such as XML Schema validation, XML transformation,
XML signatures, and XML encryption

  THENuse XML gateway

OTHERWISE IF cross-domain security mediation OR robust security protection is required

  THENuse XSM or a combination of XSM and XML gateway

OTHERWISE IF complex message transformations are required OR protocol transformations are required OR
communication style mediation is required

  THENuse ESB

OTHERWISE IF XSM is being used for operational monitoring, SLA compliance monitoring, error tracking,
root-cause analysis, BAM, or other monitoring purposes AND additional basic mediation is required

  THENuse XSM

OTHERWISE IF dynamic service selection or load balancing or content-based routing or simple message
transformations are required

  THENuse XSM or XML gateway or ESB

OTHERWISE IF endpoint policy enforcement is required to support basic mediation capabilities, such as
validation, transformations, versioning, compression, logging, authentication, authorization, auditing, encryption,
and signature processing

  THEN IF the infrastructure comprises multiple platforms
        THENuse XSM
  OTHERWISEuse platform interceptor

OTHERWISEdo not use any mediation services
```

Alternative Service Mediation position statements (important: choose *all* that apply):

Keep the number of intermediaries to a minimum.

Each intermediary in a message path increases the latency of a message interaction. When selecting and configuring intermediaries, always consider the performance impact they will have on the application. Whenever possible, combine mediation functionality into a single intermediary.

And…

Use XML gateway.

XML gateways provide hardware-based acceleration of XML processing, and they are more effective at supporting performance and scalability issues than software-based solutions.

Or…

Use XSM or a combination of XSM and XML gateway.

XSM and XML gateway solutions support credential mapping and cross-domain security mediation. In situations where services require more robust security protections, organizations should employ a combination of transport-level and application-level security protections, as well as layered defenses based on PEPs deployed at centralized, intermediary, and endpoint locations. XSM supports all three PEP topology options, and provides robust security features. An XML gateway offers accelerated processing of XML encryption and XML signatures. It also supports content filtering and intrusion detection. If these features are desired, organizations should consider using an XML gateway in conjunction with XSM. For more information about SOA security strategies, see the *Application Platform Strategies* MBP document, "Developing a Web Services Security Strategy."

Or…

Use ESB.

ESBs are designed to support semantic mediation, such as complex message transformations, protocol transformations, and communication style mediation. Simple message transformations can be performed by any mediation system using XSL Transformations (XSLT) or XML Query Language (XQuery), but complex mediations that involve application logic, data aggregation, database lookups, or rule-based filtering are best performed by an ESB. ESBs can also intermediate between applications that communicate using different communication styles, such as synchronous request/response and asynchronous queuing. In addition, ESBs specialize in intermediating between various communication protocols.

Or…

Use XSM.

If XSM is already being used to monitor services, the same XSM agents can and should be used to implement basic mediation functions for the reasons stated in the "keep the number of intermediaries to a minimum" basis of position. If the SOA runtime infrastructure comprises multiple platforms, organizations should use XSM to implement endpoint PEPs to reduce management and administrative efforts as stated in the "use built-in administration tools to administer services and use XSM to administer and enforce service policies" basis of position.

Or…

Use XSM or XML gateway or ESB.

All mediation systems that support an intermediary PEP topology (XSM, XML gateway, or ESB) can perform basic mediation functions such as dynamic service location, load balancing, content-based routing, and simple message transformations using XSLT or XQuery. Organizations should configure these mediation functions to be executed by other mediation services already deployed into the message path if they exist.

Or…

Use platform interceptor.

If the SOA runtime infrastructure comprises only a single platform, the XSP provides native support for basic endpoint mediation and a single administrative environment. If the organization later adopts additional platforms, it should consider adopting XSM to manage service policies.

Or…

Do not use any mediation services

Each intermediary imposes some overhead; therefore, if mediation services are not required, then organizations should not use them.

Service Registry Position

What type of service registry should be used?

The logic for choosing a service registry selection position is as follows:

IF an organization is in the early stage of SOA adoption, has a small number of developers, and has not yet deployed more than a dozen services

THENuse an informal registry

OTHERWISE IF an organization implements a SOA runtime infrastructure using only products from a single vendor and that vendor supplies a platform-specific registry

THENuse a platform-specific registry

OTHERWISE IF an organization uses ebXML middleware for B2B integration

THENuse a combination of ebXML and UDDI

OTHERWISEuse UDDI

Alternative Service Registry position statements (important: choose only *one*):

Use an informal registry.

During the early stages of SOA adoption, the organization can rely on human-oriented communication and collaboration systems, such as a wiki or spreadsheet, to exchange information about services. As service-oriented systems grow in sophistication, though, this method of communication cannot scale, nor can it support the needs of an MCI. The components in the SOA runtime infrastructure must exchange information, and to do so, they must use standard data models and protocols.

Or…

Use a platform-specific registry.

IBM and IONA are two vendors that supply platform-specific registries that complement their SOA platform offerings (WebSphere and Artix, respectively). In both cases, the platform products and tools are integrated with the platform-specific registry. The platform-specific registry significantly enhances the capabilities of each individual platform component and provides cohesion to the platform suite as a whole. The platform products are not as well integrated with UDDI-compliant registries as they are with the platform-specific registries. Unfortunately, third party SOA runtime infrastructure products cannot interface with the platform-specific registries. Organizations that exclusively use either WebSphere or Artix products will benefit from using the associated platform-specific registry, but these registries are inappropriate for organizations that use a multivendor SOA runtime infrastructure. See the *Application Platform Strategies* Product Profile document, " IBM WebSphere Service Registry and Repository," for more information.

Organizations that use WebSphere or Artix in a multivendor environment should consider using the platform-specific registry in conjunction with a UDDI registry.

Or…

Use a combination of ebXML and UDDI.

The ebXML registry model was designed specifically to support B2B integration using the ebXML middleware framework. Organizations that use ebXML extensively will find it more convenient to use an ebXML registry than a UDDI registry to support these applications. But WSF-based design, development, and management tools don't support the ebXML protocols. Therefore, an organization should also use a UDDI-compliant registry to support WSF-based interactions.

Or…

Use UDDI.

UDDI is broadly adopted in numerous types of SOA infrastructure products, such as development tools, service platforms, service mediation, and service management systems. UDDI provides an essential interoperability protocol that enables infrastructure components to work together. For widest interoperability, the registry should support UDDI v2. For greater features and functionality, especially in regards to governance and lifecycle management, the registry should support UDDI v3 and the optional UDDI v3 Subscription API. All UDDI v3-compliant registries are required to support the UDDI v2 protocols; therefore, UDDI v3 is a better choice.

Service Administration Position

What types of administration systems should be used to configure services and service policies?

The logic for choosing a service administration position is as follows:

IF all services within an organization are hosted by a single platform AND the platform provides appropriate built-in security and integrity capabilities

THENuse the XSP's built-in administration tools to administer services and service policies

OTHERWISEuse built-in administration tools to administer services and use XSM to administer and enforce service policies

Alternative Service Administration position statements (important: choose only *one*):

Use the XSP's built-in administration tools to administer services and service policies.

XSPs provide proprietary built-in administration tools for configuring services hosted within the XSP. Most XSPs also provide proprietary administration tools for configuring security and integrity policies and interceptors. It is appropriate to use these proprietary tools in a homogeneous environment.

Or…

Use built-in administration tools to administer services and use XSM to administer and enforce service policies.

Because each XSP uses proprietary administration tools, any organization that uses multiple XSPs should adopt an XSM solution in order to gain control over policy management and enforcement in the diverse environment. (Unfortunately, there's no alternative to using the XSP built-in service configuration tools.) In situations where the XSP does not provide adequate security and integrity capabilities, the XSM solution can provide these capabilities.

Service Monitoring Position

What types of monitoring systems should be used to provide visibility into the SOA environment?

The logic for choosing a service monitoring position is as follows:

IF the organization uses a single enterprise systems management (ESM) solution AND the ESM vendor provides platform-independent SOA monitoring capabilities AND a XSM solution is not being used for mediation or administration purposes

THENuse an ESM SOA monitoring solution

OTHERWISEuse XSM

Alternative Service Monitoring position statements (important: choose only *one*):

Use an ESM SOA monitoring solution.

The leading ESM vendors (CA, Hewlett-Packard, and IBM) now provide SOA monitoring solutions that support SLA compliance tracking, error management, and root-cause analysis. (Note that IBM's SOA monitoring solution, ITCAM for SOA, currently supports a very limited set of XSPs.) These SOA management solutions are well integrated with their associated ESM solutions. For organizations that have standardized on a single ESM solution and that have not adopted XSM for other reasons, an ESM-based solution is more appropriate.

Or…

Use XSM.

The ESM vendors' SOA monitoring solutions are designed to work as extensions to the ESM frameworks and are less effective as stand-alone monitoring products. Any organization that is already using an XSM solution for service mediation, policy configuration, or policy enforcement should use the XSM solution for monitoring rather than using an additional ESM solution. As indicated in the “Service Mediation” position, the number of intermediaries in a message path should be kept to a minimum.

XSM provides comprehensive operational monitoring, SLA compliance tracking, error tracking, root-cause analysis, and BAM. XSMs provide dashboards for observing SOA performance. XSM can also propagate management information to traditional system management frameworks.

Relationship to Other Components

This Technical Position depends on the following components in the *Application Platform Strategies* Reference Architecture for a number of closely related topics:

- The "SOA Governance Infrastructure" Technical Position provides guidance for selecting infrastructure components that support a SOA governance program.
- The "Application Factoring" Technical Position provides guidance for how to factor application functionality into reusable services.
- The "Application Platform Foundations" Technical Position provides guidance for selecting an application platform.
- The "Application Security" Technical Position provides guidance for implementing security protections in applications and services.
- The Root Template defines a conceptual model for a SOA infrastructure.
- The "Managed Communications Infrastructure" Template defines a functional model of the components that make up a SOA runtime infrastructure.
- The "SOA Governance Infrastructure" Template defines a functional model of the infrastructure components that support a SOA governance program.
- The "Infrastructure Services Model" Template defines a model for externalizing infrastructure functionality into a set of shared services.
- The "Web Services Framework Standards" Template explains the composable architecture of the WSF.

This Technical Position also references the following documents:

- *Application Platform Strategies* Root Document, "Turning the Network Into the Computer: The Emerging Network Application Platform"
- *Application Platform Strategies* overview, "Service Oriented Architecture: Developing the Enterprise Roadmap "
- *Application Platform Strategies* overview, "VantagePoint 2005–2006 SOA Reality Check"
- *Application Platform Strategies* overview, "Enterprise Architects: Sowing the Seeds of SOA Success"
- *Application Platform Strategies* MBP document, "Building the Business Case for Service Oriented Architecture Investment"
- *Application Platform Strategies* overview, "The Advent of the Network Platform: Web Services Move into the IT Fabric"
- *Application Platform Strategies* MBP document, "Developing a Web Services Security Strategy"
- *Application Platform Strategies* report, "Web Services Security: A Plethora of Products"
- *Identity and Privacy Strategies* Reference Architecture Technical Positions, "User Authentication," "User Authorization," "Identity Lifecycle Management," "Federated Identity," "Public Key Infrastructure," and " Roles"
- *Security and Risk Management Strategies* Reference Architecture Technical Positions, "System Placement," " Encryption," "Perimeters and Zones," "Vulnerability Management," and "Technical Security Policy Management: Mapping Intent into Implementation"
- *Application Platform Strategies* subthread, "Web Services Management"
- *Application Platform Strategies* Technology & Standards document, "Web Services Management"
- *Application Platform Strategies* Market Landscape document, "Web Services Management: Townsman of a Stiller Town"
- *Application Platform Strategies* Product Profile document, "AmberPoint's SOA Management System and SOA Validation System"
- *Application Platform Strategies* Product Profile document, "Progress Software's Actional 6.0"
- *Application Platform Strategies* Product Profile document, "SOA Software's Service Manager 3.1"

- *Application Platform Strategies* report, "Scaling and Accelerating Web Services: The Roadmap to High Performance"
- *Application Platform Strategies* overview, "Standardizing EAI with Web Services: An Inevitable But Prolonged Transition"
- *Application Platform Strategies* overview, "Crossing the Divide: The Mechanics of Process Execution"
- *Application Platform Strategies* report, "Enterprise Service Bus: EAI in Transition"
- *Application Platform Strategies* report, "Integration Brokers: Providing the Middleware Fabric for Complex Integration"
- *Security and Risk Management Strategies* overview, "Web Services Security Standards 2006: Where Are We Now?"
- *Application Platform Strategies* report, "Extensible Markup Language (XML) 2005: Core XML Standards and Their Impact on Business Development"
- *Application Platform Strategies* report, "The P-Languages: PHP, Perl, and Python for Enterprise Scripting"
- *Application Platform Strategies* report, "Ruby on Rails"
- *Collaboration and Content Strategies* overview, "Transforming Communication Channels: RSS Within the Enterprise"
- *Application Platform Strategies* overview, "The World Wide Web: An Introduction"
- *Application Platform Strategies* report, "The Java Superplatforms: Comparing IBM, Oracle, and BEA"
- *Application Platform Strategies* report, "The Microsoft Superplatform: Setting the Bar in the Superplatform Arms Race"
- *Application Platform Strategies* report, "SAP NetWeaver: Potential Wildcard in the Superplatforms Arms Race "
- *Application Platform Strategies* report, "The JBoss Enterprise Middleware Suite"
- *Application Platform Strategies* report, "JEE5: The Beginning of the End of Java EE"
- *Application Platform Strategies* overview, "Java Standards for Web Services: Bringing Synergy to a Fractured Environment"
- *Application Platform Strategies* overview, "Selecting a Java Web Services Platform: An Evaluation Framework "
- *Application Platform Strategies* Technology & Standards document, "Registry Services: The Foundation for SOA Governance"
- *Application Platform Strategies* Market Landscape document, "The Registry and SOA Governance Market Landscape"
- *Application Platform Strategies* Product Profile document, "HP SOA Systinet"
- *Application Platform Strategies* Product Profile document, " IBM WebSphere Service Registry and Repository "
- *Application Platform Strategies* Product Profile document, "Microsoft Enterprise UDDI Services"
- *Application Platform Strategies* Product Profile document, "webMethods Infravio X-Registry"

Revision History

May 2007

- Updates to the Technical Position to reflect the recent changes made to the Reference Architecture Templates.
- Refactored the governance infrastructure discussion into a separate Technical Position (“SOA Governance Infrastructure”).
- Softened the language regarding the recommendation to use WSF as the foundation for SOA.
- POX now treated as a serious alternative.
- Adopted new terminology: web services platform (WSP) and web services management (WSM) are now referred to as XML services platform (XSP) and XML services management (XSM).
- Shifted orchestration engine from a mediation alternative to a platform alternative.
- Expanded the discussion of mediation alternatives, and added a matrix comparing alternatives with capabilities.
- Added platform interceptor as a mediation alternative.
- Refactored the service management component into service administration and service monitoring components.
- Refocused registry discussion just on runtime capabilities.
- Defined a set of alternatives for registry (UDDI, ebXML, platform-specific, and informal).

January 2006

- This is the initial iteration of this Technical Position. It replaces the Reference Architecture Technical Position, “Web Services Integration.”

Author Bio

Anne Thomas Manes

Vice President and Research Director

Emphasis: Service-oriented architecture, web services, XML, governance, superplatforms, application servers, Java, J2EE, .NET, application security, data management

Background: Former Chief Technology Officer at Systinet, a SOA governance vendor (now part of HP). Director of Market Innovation in Sun Microsystem's software group. Manes' 28-year industry background also includes field service and education at IBM Corporation; customer education at Cullinet Software; product management at Digital Equipment Corporation; chief architect at Open Environment Corporation; and research analyst with Patricia Seybold Group.

Primary Distinctions: Named one of the 50 most powerful people in networking in 2002 by Network World. Listed among the "Power 100 IT Leaders," by Enterprise Systems Journal. A frequent speaker at trade shows and InfoWorld, JavaOne, and RSA conferences. She has also authored numerous articles in trade publications. Member of Web Services Journal editorial board. Authored "Web Services: A Manager's Guide," published by Addison-Wesley, 2003. Participated in web services standards development efforts at W3C, OASIS, WS-I, and JCP. Anne earned a BA in Economics at Wellesley College.