Web Services architecture overview
The next stage of evolution for e-business

IBM Services Architecture Team

September 06, 2000

This article presents an overview of IBM Web Services architecture, including what Web Services are, the fundamental characteristics of a Web Services architecture and the benefits of this approach.

- Web Services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the World Wide Web.
- The Web Services architecture describes three roles: service provider, service requester and service broker; and three basic operations: publish, find and bind. A network component can play any or all of these roles.
- Two separate documents describe Web Services: A Well-Defined Service (WDS) document describes nonoperational service information, such as service category, service description, and expiration date, as well as business information about the service provider, such as company name, address, and contact information. A Network-Accessible Service Specification Language (NASSL) document describes operational information about the service, such as service interface, implementation details, access protocol, and contact endpoints.
- A Web Services architecture implementation should allow for incremental security and quality of service models facilitated by configuring a set of environmental prerequisites (for example, authentication mechanism, billing, and so on) to control and manage the interactions.
- Web Services can be dynamically composed into applications stemming from capabilities-based look-up at runtime, instead of the traditional static binding. The dynamic nature of the collaborations allow the implementations to be platform- and programming language-neutral, and communications mechanism-independent, while creating innovative products, processes, and value chains.

Introduction

Previous attempts at distributed computing (CORBA, Distributed Smalltalk, Java RMI) have yielded systems where the coupling between various components in a system is too tight to be effective for low-overhead, ubiquitous B2B e-business over the Internet. These approaches require too much agreement and shared context among business systems from different organizations to be reliable for open, low-overhead B2B e-business.
Meanwhile, the current trend in the application space is moving away from tightly coupled monolithic systems and towards systems of loosely coupled, dynamically bound components. Systems built with these principles are more likely to dominate the next generation of e-business systems, with flexibility being the overriding characteristic of their success. **We believe that applications will be based on compositions of services discovered and marshaled dynamically at runtime (just-in-time integration of services).** Service (application) integration becomes the innovation of the next generation of e-business, as businesses move more of their existing IT applications to the Web, taking advantage of e-portals and e-marketplaces and leveraging new technologies, such as XML.

The concept of Web Services, described here, is our view of what the next generation of e-business architectures for the Web will look like. The Web Services architecture describes principles for creating dynamic, loosely coupled systems based on services, but no single implementation. There are many ways to instantiate a Web Service by choosing various implementation techniques for the roles, operations, and so on described by the Web Services architecture.

Various environmental aspects must also be considered when designing Web Services. For example, the security requirements for services brokers will vary depending upon the deployment environment. Most intranet deployments have minimal security requirements but in situations where high-value B2B transactions are conducted, much higher security may be necessary. An approach is to take a risk-assessment view of security and design brokers to provide different levels of information based upon an environment's security infrastructure. (For more information about Web Services security considerations, see the Appendix.) Future releases of the Web Services toolkit will support some or all of these security considerations.

**A new architectural approach**

Traditional systems architectures incorporate relatively brittle coupling between various components in the system. The bulk of IT systems, including Web-oriented systems, can be characterized as tightly coupled applications and subsystems. IBM CICS transactions, databases, reports, and so on are built with tight coupling, using data structures (database records, flat files).

Monolithic systems like these are sensitive to change. A change in the output of one of the subsystems will often cause the whole system to break. A switch to a new implementation of a subsystem will also often cause old, statically bound collaborations (which unintentionally relied on the side effects of the old implementation) to break down. This situation is manageable to a certain extent through skills and numbers of people. As scale, demand, volume, and rate of business change increase, this brittleness becomes exposed. Any significant change in any one of these aspects will cause the brittleness of the systems to become a crisis: unavailable or unresponsive Web sites, lack of speed to market with new products and services, inability to rapidly shift to new business opportunities, or competitive threats. IT organizations will not be able to cope with changes because of the coupling; the dynamics of the Web makes management of these brittle architectures untenable.
We need to replace the current models of application design with a more flexible architecture, yielding systems that are more amenable to change.

**Web Services**

**Just-in-time integration**

The Web Services architecture describes the principles behind the next generation of e-business architectures, presenting a logical evolution from object-oriented systems to systems of services. Web Services systems promote significant decoupling and dynamic binding of components: All components in a system are services, in that they encapsulate behavior and publish a messaging API to other collaborating components on the network. Services are marshaled by applications using service discovery for dynamic binding of collaborations. Web Services reflect a new service-oriented architectural approach, based on the notion of building applications by discovering and orchestrating network-available services, or just-in-time integration of applications.

With the Web Services approach, application design becomes the act of describing the capabilities of network services to perform a function and describing the orchestration of these collaborators. At runtime, application execution is a matter of translating the collaborator requirements into input for a discovery mechanism, locating a collaborator capable of providing the right service and orchestrating message sends to collaborators to invoke their services. These new applications, themselves, become services, thus creating aggregated services available for discovery and collaboration.

**What are Web Services?**

Web Services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the Web.

The Web Services architecture is the logical evolution of object-oriented analysis and design, and the logical evolution of components geared towards the architecture, design, implementation, and deployment of e-business solutions. Both approaches have been proven in dealing with the complexity of large systems. As in object-oriented systems, some of the fundamental concepts in Web Services are encapsulation, message passing, dynamic binding, and service description and querying. Fundamental to Web Services, then, is the notion that everything is a service, publishing an API for use by other services on the network and encapsulating implementation details.

**Web Services components**

Several essential activities need to happen in any service-oriented environment:

1. A Web service needs to be created, and its interfaces and invocation methods must be defined.
2. A Web service needs to be published to one or more intranet or Internet repositories for potential users to locate.
3. A Web service needs to be located to be invoked by potential users.
4. A Web service needs to be invoked to be of any benefit.
5. A Web service may need to be unpublished when it is no longer available or needed.
A Web Services architecture then requires three fundamental operations: publish, find, and bind. Service providers publish services to a service broker. Service requesters find required services using a service broker and bind to them. These ideas are shown in the following figure.

**Publish, find, and bind**

![Publish, find, and bind diagram]

The mechanism of service description is one of the key elements in a Web Services architecture. The full description of a service for our architecture is found in two separate documents, a Network Accessible Service Specification Language (NASSL) document and a Well-Defined Service (WDS) document. NASSL is an XML-based Interface Definition Language (IDL) for network-based services, and is used to specify the operational information for a Web Service, such as service interface, implementation details, access protocol, and contact endpoints. A WDS document is used to specify the nonoperational information for a service, such as service category, service description, and expiration date, as well as business information about the service provider, such as company name, address, and contact information. A WDS document is complementary to a corresponding NASSL document. Together these two documents are used to specify a full service description that allows service requesters to locate and invoke a service.

All collaborations in the Web Services architecture have the possibility of being controlled by a configurable, negotiable set of environmental prerequisites. An environmental prerequisite is any nonfunctional component or infrastructure mechanism that must be made operational before a service can be invoked -- for example, the use of a particular communications mechanism (HTTPS, IBM MQSeries), or the use of a particular third-party auditing or billing service. These components (often, themselves, implemented as services) must be put in place before the service can actually be invoked.

A service may support multiple possible implementations for any environmental prerequisite it specifies. For example, the service could offer a choice of communications layer, choice of billing service, or other option. The service requester can then negotiate or choose which implementation to use to satisfy the environmental prerequisite. It is through environmental prerequisites that collaborations can be as secure, reliable, and safe as required by the two collaborators in a Web Services architecture.

**Web Services benefits**

Use of the Web Services architecture provides the following benefits:
• **Promotes interoperability by minimizing the requirements for shared understanding**
XML-based interface definition language (NASSL), an XML-based service description
(WDS) and a protocol of collaboration and negotiation are the only requirements for shared
understanding between a service provider and a service requester. By limiting what is
absolutely required for interoperability, collaborating Web services can be truly platform
and language independent. By limiting what is absolutely required, Web Services can be
implemented using a large number of different underlying infrastructures.

• **Enables just-in-time integration**
Collaborations in Web Services are bound dynamically at runtime. A service requester
describes the capabilities of the service required and uses the service broker infrastructure
to find an appropriate service. Once a service with the required capabilities is found, the
information from the service's NASSL document is used to bind to it.
Dynamic service discovery and invocation (publish, find, bind) and message-oriented
collaboration yield applications with looser coupling, enabling just-in-time integration of new
applications and services. This in turn yields systems that are self-configuring, adaptive and
robust with fewer single points of failure.

• **Reduces complexity by encapsulation**
All components in Web Services are services. What is important is the type of behavior a
service provides, not how it is implemented. A WDS document is the mechanism to describe
the behavior encapsulated by a service.
Encapsulation is key to:
  • Coping with complexity. System complexity is reduced when application designers do not
    have to worry about implementation details of the services they are invoking.
  • Flexibility and scalability. Substitution of different implementation of the same type of
    service, or multiple equivalent services, is possible at runtime.
  • Extensibility. Behavior is encapsulated and extended by providing new services with
    similar service descriptions.

• **Enables interoperability of legacy applications**
By allowing legacy applications to be wrappered in NASSL and WDS documents, and
exposed as services, the Web Services architecture easily enables new interoperability
between these applications. In addition, security, middleware and communications
technologies can be wrappered to participate in a Web service as environmental
prerequisites. Directory technologies, such as LDAP, can be wrappered to act as a service
broker.
Through wrappering the underlying plumbing (communications layer, for example), services
insulate the application programmer from the lower layers of the programming stack. This
allows services to enable virtual enterprises to link their heterogeneous systems as required
(through http-based communications) and/or to participate in single, administrative domain
situations, where other communications mechanisms (for example, MQSeries) can provide a
richer level of functionality.
Examples of this can be found in merger situations, where the resulting enterprise must
integrate disparate IT systems and business processes. A service-oriented architecture
would greatly facilitate a seamless integration between these systems. Another example can
be found in the combination of the travel industry with pervasive computing, when largely
mainframe-based travel applications can be exposed as services through wrappering and made available for use by various devices in a service-oriented environment. New services can be created and dynamically published and discovered without disrupting the existing environment.

Summary

Web Services is the next stage of evolution for e-business -- the result of viewing systems from a perspective that everything is a service, dynamically discovered and orchestrated, using messaging on the network.

In the Web Services architecture, each component is regarded as a service, encapsulating behavior and providing the behavior through an API available for invocation over a network. This is the logical evolution of object-oriented techniques (encapsulation, messaging, dynamic binding, and reflection) to e-business.

The fundamental roles in Web Services are service providers, service requesters, and service brokers. These roles have operations: publish, find, and bind. Operation intermediation occurs through environmental prerequisites, and it introduces aspects such as security, workflow, transactions, billing, quality-of-service, and service level agreements. The mechanism of service description language is key to fundamental operations in Web Services. A complete description of a Web Service appears in two separate documents: a Network-Accessible Service Specification Language (NASSL) document and a Well-Defined Service (WDS) document.

The Web Services architecture provides several benefits, including:

- Promoting interoperability by minimizing the requirements for shared understanding
- Enabling just-in-time integration
- Reducing complexity by encapsulation
- Enabling interoperability of legacy applications

Appendix: Security and trust issues

Security for a Web Services architecture will need to propose a shift in the traditional understanding of security to accommodate new models for supporting the finding and execution of services in an open dynamic Web environment. The security goal of a service-oriented architecture is to enable trusted interactions among the roles. If security is defined as protection against threats, a Web Service will identify its set of perceived threats and propose methods of preventing threats to Web Service interactions. Trust can be established between two parties when they understand their risks, having identified the threats and vulnerabilities and agreed on a set of countermeasures and safeguards for protecting themselves in doing business.

One of the difficulties in capturing threats and countermeasures is that they interact in a cycle in which new threats are found, leading to the creation of a countermeasure which may lead to a new threat. Therefore, some countermeasure definition must be based on speculation. We cannot wait for the evolution of all possible threats and countermeasures to be identified before attempting
to address security in the systems we design. We approach security for Web Services-oriented architectures as a challenge to understand and document the existing threats and proposed countermeasures. The architecture then attempts to capture the thinking behind the current conditions and to allow for flexible implementation of different levels of security mechanisms, having control points for integration with existing middleware and new security mechanisms. The evolution of security measures is important to the evolution of the new economy, but we have experience to show that, to be successful, any implementation must be simple to design, simple to use, ubiquitous and cost-effective, or people will not use it. And, hence, it will be ineffective at best.

We have identified several areas of threats that need to be addressed with countermeasures:

- The security of information that is shared between the broker, the requester, and the provider at runtime
- The security of the network in which the runtime is deployed
- The security of the programming model (APIs, skeletons, stubs, and so on) at design time.

In addition to the issues of application security models at design time, another set of security issues exists for the application development environment itself. This creates a tension between the runtime and the design-time components in the area of security. It is also a requirement that Web Services be simple, be based on open standards, and be extensible. These three requirements resonate with basic security goals for any new development. The difficulty arises in trying to incorporate legacy security programming models and mechanisms as part of a new design.

Access control decisions for services and delegation models must exist and ultimately be enforced within the application. The tools are in the application development environment and are not part of the Web Services architecture, but will be part of the larger B2B environment. Tooling may be able to facilitate some of the design-time issues adding access control logic to applications, but may not be able to address all legacy conditions. We recommend that service providers use the prerequisites section to communicate to the service requester any information that needs to be supplied by the requester to qualify for access control decisions.

There are several possible access control models when designing brokers:

A **promiscuous broker** is one in which there is no checking for authentication of requesters or publisher. The broker provides public access to the information in its data repository and makes no claim to the data being correct. The threats to this type of implementation include impersonation of the service requester, impersonation of the service provider, unauthorized modification of data, disclosure of information, denial of service, and repudiation of action.

An **authenticated broker** is one that authenticates both requesters and publishers. This allows the broker to make informed decisions about who is accessing the information and the broker is capable of enforcing access control on the data it keeps. Threats to this type of implementation include impersonation of the requester, impersonation of the service provider, unauthorized modification of data, disclosure of information, denial of service, and repudiation of action. This type of broker would implement the requirement that requests for services be invoked over the
HTTPS protocol, using SSL to enforce authentication of entities. It could also integrate parser support for XMLDSIG to validate signing the XML documents defined to be part of the message exchange.

A fully authorized broker is one who would implement an authorization paradigm and store access information with each data entry. It would establish ownership of entries and enforce the requirement that only authorized agents are able to modify data. It could also implement a more granular authorization engine that would allow only certain subsets of requesters to access particular information. Threats to this type of implementation include impersonation of the requester, impersonation of the service provider, unauthorized modification of data, disclosure of information, denial of service, and repudiation of action. This type of broker would offer its services over HTTPS, using SSL to enforce authentication of entities with parser support for XMLDSIG to sign the XML documents defined to be part of the message exchange; and middleware infrastructure, like Tivoli/Dascom, and the AZN API to enforce enterprise-wide access control decisions.
Related topics

- Get more information about the UDDI project at www.uddi.org.
- Web Services Toolkit Version 1.2 features:
  - Support for IBM WebSphere Application Server Version 3.5
  - Support for Linux
  - Migration to Apache Xerces 1.1.3 and SOAP 2.0
  - Improved documentation

This updated version also includes:
- Support for Web Services Description Language (WSDL)
- A subset of the Universal Discovery Description & Integration (UDDI) APIs
- Support for Advertisement and Discovery of Services (ADS)
- A new Web services creation tool.

© Copyright IBM Corporation 2000
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)