WEB SERVICES

What Are Web Services?

Simply put, Web Services are loosely coupled, contracted components that communicate via XML-based interfaces. Let's take a closer look at this definition:

- Loosely coupled means that Web Services and the programs that invoke them can be changed independently of each other. Loose coupling also implies that Web Services are platform independent.
- Contracted means that a Web Service's behavior, its input and output parameters, and how to bind to it are publicly available.
- A component is encapsulated code, which means that the implementation of each component is hidden from outside the component. Each component's functionality is only known by the interface it exposes.
- Because all Web Services' interfaces are built with XML, they all share the advan- tages of XML: They have a human readable, text-based format that is firewall friendly and self-describing. All Web Services are described using a standard XML notation called its service description.

Put another way, Web Services are self-contained applications that can be described, published, located, and invoked over the Internet (or any network, for that matter).

Business Motivations for Web Services;

The vision of global e-business largely remains unrealized. Executives dream about seamless interactions both with other companies as well as e-marketplaces, but the technology lags behind the vision. Today's information technology is still extraordinarily complex and expensive. Even with standards such as Electronic Data Interchange (EDI), Java 2 Enterprise Edition (J2EE), Common Object Request Broker Architecture (CORBA), and Windows Distributed interNet Application (Windows DNA), communicating between different corporate systems is still filled with hair-pulling detail work.

The business world needs more powerful techniques to scale business solutions without increasing complexity to unmanageable levels. In addition, there is a clear need for open, flexible, and dynamic solutions for enabling global e-business interactions among systems. The Web Services model promises to deliver these solutions by addressing complexity and costs, providing a common language for B2B e-commerce, and enabling the vision of a global e-marketplace.

B2B E-Commerce:

Business to Business (B2B) e-commerce has been around for more than a decade in the form of the Electronic Data Interchange (EDI). EDI is quite powerful and has gained widespread acceptance but is limited by its semantic ambiguity. For example, a "quantity" field in a given

form may stand for number of boxes for one company but the number of pallets for another. People have to resolve each ambiguity manually, making EDI useful primarily in a hub-and-spoke arrangement, where one large company can dictate the meaning of each field to its suppliers.

When the Internet opened up the prospect of many-to-many e-commerce, it soon became clear that there needed to be a way to agree upon a single business vocabulary for all participants in each trading group. XML provided the basis for building such vocabularies because of its inherent extensibility. However, XML's greatest strength also proved to be a weakness, because its extensibility led to hundreds of different business vocabularies, often with overlapping applicability.

The Web Services model addresses this Tower of Babel problem by providing for dynamic service descriptions. Individual Web Services can describe their interfaces at runtime, allowing for dynamic interpretation of the semantics of the XML that underlies the messages Web Services send and receive.

Technical Motivations for Web Services:

The technical motivations for Web Services are far more complex than the business motivations. Fundamentally, technologists are looking for the simplicity and flexibility promised, but never delivered, by RPC architectures and object-oriented technologies.

Limitations of CORBA and DCOM:

Programming has been performed on a computer-by-computer basis for much of the history of computing. Programs were discrete chunks of computer code that ran on individual computers. Even object-oriented programming originated in a single-computer environment. This isolated computer mind set has been around so long that it pervades all thinking about software.

Then along came networks, and technologists looked for ways to break up program functionality onto multiple computers. Early communication protocols, such as the Network

File System for Unix and Microsoft's Distributed Computing Environment, focused on the network layer. These protocols, in turn, led to the development of wire protocols for distributed computing—in particular, the Object Remote Procedure Call (ORPC) proto- col for Microsoft's DCOM and the Object Management Group's Internet Inter-ORB Protocol (IIOP) that underlies CORBA.

RPC architectures such as DCOM and CORBA enabled programs to be broken into different pieces running on different computers. Object-oriented techniques were particularly suited to this distributed environment for a few reasons. First, objects maintained their own discrete identities. Second, the code that handles the communication between objects could be encapsulated into its own set of classes so that programmers working in a distributed environment needn't worry about how this communication worked.

However, programmers still had that isolated computer mindset, which colored both DCOM's and CORBA's approach: Write your programs so that the remote computer appears to be a part of your own computer. RPC architectures all involved marshalling a piece of a program on one computer and shipping it to another system.