Introduction

In the early days of the Internet, Web applications delivered static webpages via HTML. Certainly, the development of websites was simpler; however, static content can quickly become outdated; thus, the content management of a website is important.

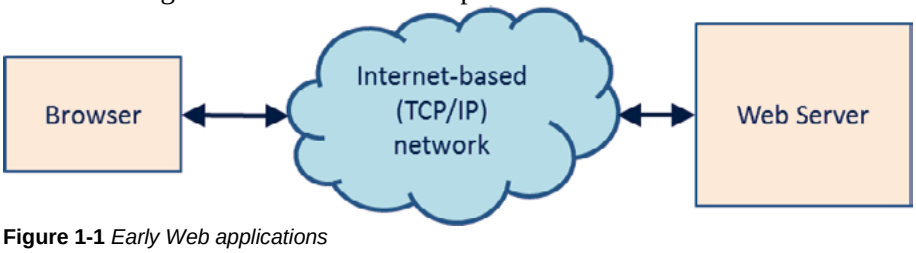

Figure 1-1 *Early Web applications*

In order to provide dynamic content to Web users, 2-tier web applications were realized with the introduction of the Common Gateway Interface (CGI), which retrieves content from external data resources, such as a database. CGI acts as a client in the traditional client-server architecture. A CGI script processes the request and returns the result to the Web server. The server then formats the contents in HTML and returns to the browser for display.

CGI suffered many drawbacks that necessitated changes to the 2-tier architecture. The database was often running on the same machine; therefore, making backups of the data was difficult. CGI was running as a separate process, so it suffered from a context-switching penalty. CGI was not designed for performance, security or scalability.

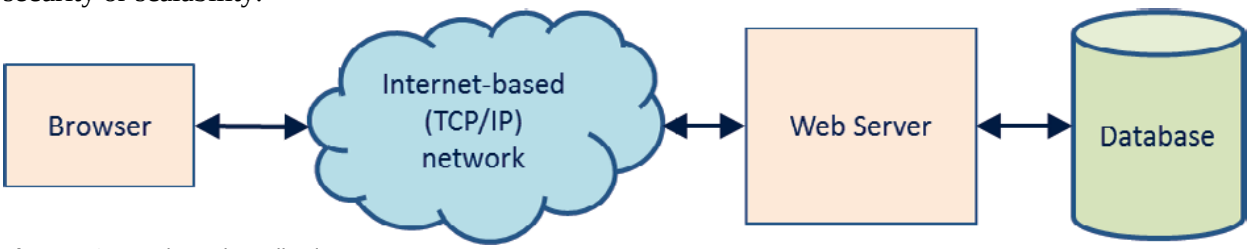

Figure 1-2 *Two-tier web application*

Nowadays, n-tier Web application architecture is commonly used. In this architecture, middleware or an application server is introduced to connect the Web server and the database more efficiently. The performance of an n-tier application is improved because Web servers, middleware and databases can be hosted by separate machines. Each tier can be replicated for the purposes of load balancing. Security is also improved because data is not stored on the Web or application server, which makes it harder for hackers to gain access into the database where data is stored.

Figure 1-3 *An n-tier web architecture*

A web and an application servers are often run on the same machine; however, it is best practice to run the database server on a separate machine. In a software development environment, all three servers can be hosted on a single machine. In this book, a server is often referred to a software application.

1.1 Browsing the Internet

Before the conception of Web 2.0 (around 1999), the basic use of the World Wide Web (WWW) and the Internet was simple and based on the traditional client-server model with older technologies such as Remote Procedure Call (RPC) or Transaction Processing (TP) Monitors or other middleware that permitted programmable clients.

Consider a typical use case of a person browsing the Internet by means of a browser. The Web server in this example serves dynamic HTML pages using Java Server Pages (JSP) technology. In addition, it uses Enterprise Java Bean (EJB) or Plain-Old-Java-Object (POJO). JSP is oriented toward the delivery of webpages for the presentation layer. EJBs or POJOs are usually used for processing business rules. There are thousands of Web applications that use Java/JEE technology.

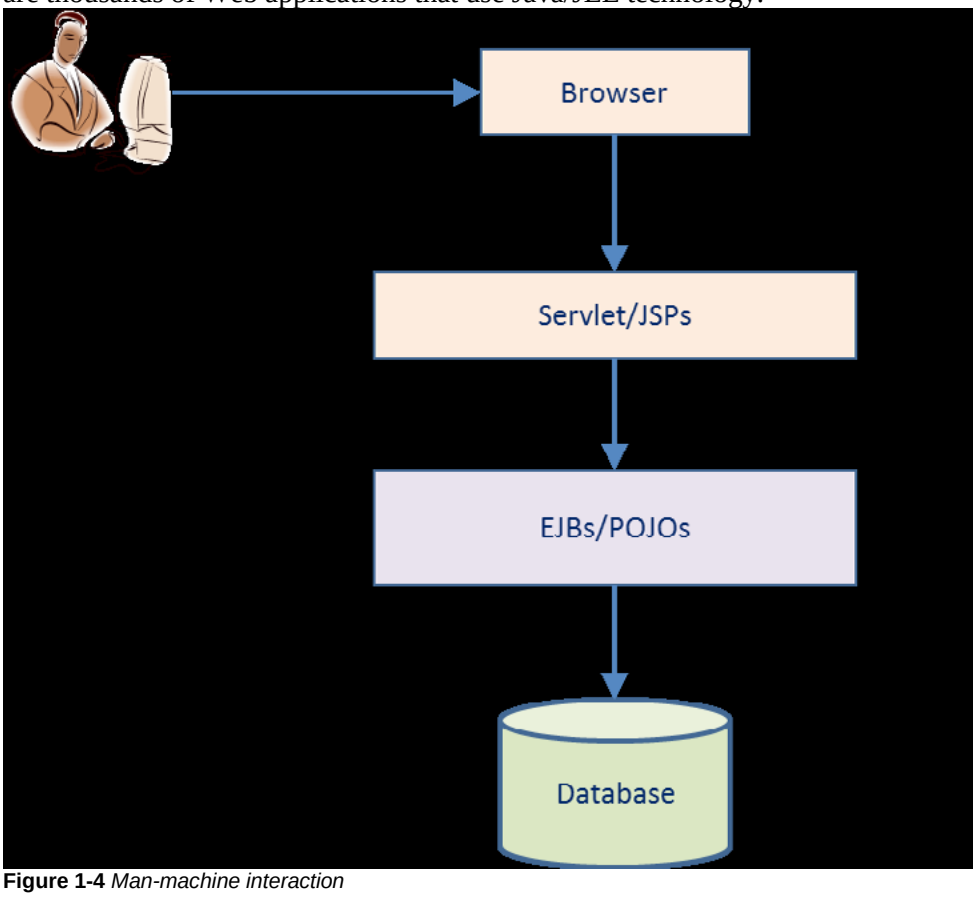

Figure 1-4 *Man-machine interaction*

The Internet architecture was originally designed for human users. HTTP protocol was for exchanging documents (Web or HTML pages). HTML was designed for basic graphical user interface (GUI) applications. Computing resources on a web browser are often idle while the user is browsing the Internet. These available resources prompted the idea of providing more robust web browsing experience. In addition, the idea of business-to-business (B2B) data exchange model also became more feasible. Accordingly, the WS architecture was introduced to support this new type of data exchange.

1.2 Web Service architecture

A service can be one of the three types of interaction: man-to-man, man-to-machine, or machine-to machine.

A restaurant service is an example of man-to-man interaction. A person withdrawing money from an Automated Teller Machine (ATM) is an example of man-to-machine interaction. Machine-to machine interaction is exemplified by a handheld device, such as a smart device (e.g., a phone or a tablet), synchronizing its address book with Microsoft Outlook. A Web Service is a type of machine-to-machine interaction that uses specific Web standards and technology. A Web Service is a set of programming interfaces, not a set of webpages.

This section begins with a basic definition of a Web Service in order to establish a basic understanding for use in later chapters. More complex aspects of Web Services will be easier to understand when the basic concept of a Web Service is properly explained.

According to W3C website, http://www.w3.org/TR/ws-desc-reqs:

A Web Service is a software application identified by a URI whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and [that] supports direct interactions with other software applications using XML based messages via Internet-based protocols.
A Web Service must involve a Web-based protocol, such as HTTP or Simple Mail Transfer Protocol (SMTP). Other transport protocols may be used, but HTTP is the most common one being used. HTTPS uses Secure Socket Layer (SSL) or Transport Secure Layer (TLS) for secured transport of data. In regard to software development concerns, the difference between HTTP and HTTPS is trivial. HTTP, thus, is used throughout this text.
A Web Service is a software application that requires interaction with another application. WS is a software integration technique for a B2B type of integration. Here, one application acts as a service provider (server) and the others act as service consumers (clients). This is a many-to-one relationship.
'Interface' is defined as "[The] point of interaction or communication between a computer and any other entity" (http://www.thefreedictionary.com). An interface can also be described as an "abstraction of a service that only defines the operations supported by the service (publicly accessible variable, procedures, or methods), but not their implementation" (Szyperski, 2002). For example, in Java, an interface can be defined and then implemented by a concrete class.
Web Service Description Language (WSDL) specifies the service interface and the rules for binding the service consumer and the provider. According to the specification of WSDL 1.1, WSDL is defined as "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information" (http://www.w3.org/TR/wsdl). WSDL defines how a consumer can interact with a service via a concrete network protocol and message format using eXtended Markup Language (XML).
XML is a profile (subset) of Standard Generalized Markup Language (SGML). SGML is a metalanguage, i.e., a language that describes other languages. Unlike HyperText Markup Language (HTML), which is used to serve static webpages, XML allows the author to create his or her own tags. Thus, XML facilitates the data and document processing functions.
Web Service relies on Simple Object Application Protocol (SOAP) as its transport. As its name implies, SOAP is a lightweight protocol that can be used to exchange structured messages (i.e., XML). SOAP 1.2 is the latest version. WSDL 1.1 supports SOAP 1.1, HTTP GET/POST, and MIME.
A service can be defined, published and discovered using some type of service registry. Current supporting service registries include electronic business XML (ebXML), Universal Discovery, Description and Integration (UDDI), and Metadata Registry (MDR). UDDI is usually a good idea; however, it is not widely used except in a private network of services.
RPC is a powerful technique that provides distributed computing capabilities across a network of machines. RPC is a form of interprocess communication that enables function calls between applications that are located across different (or the same) locations over a network. It is best suited for client-server programming.

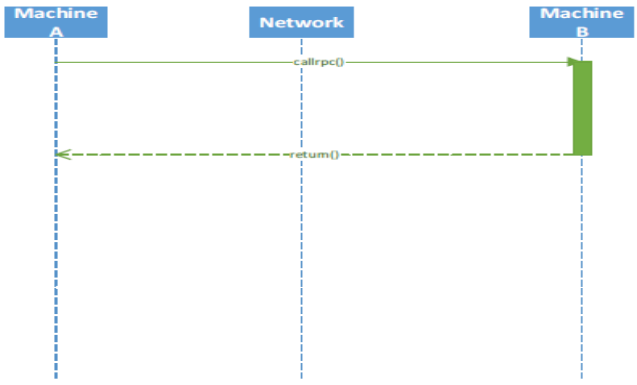

Figure 1-5 *Remote Procedure Call (RPC)*

Web Services can be used to help solve several problems in Enterprise Application Integration (EAI). Integrating existing applications for a business solution is a complex and time-consuming task. Applications that were written in different computer languages, such as C/C++, JAVA, Visual Basic, and FORTRAN, have unique logical interfaces to the external world, which makes the integration of these applications difficult, complex and time-consuming. Applications that are running on different machine architectures, such as SUN, Personal Computer, IBM Mainframes, IBM A/S 400, have unique physical interfaces to the external world. Integrating these applications is also challenging. Applications running on machines that are interconnected through a network are also difficult to integrate. The challenges of EAI arise in three main areas:

• Language barriers – XML is a standardized language that is used for message exchange
• Platform barriers – SOAP has been implemented on many platforms (e.g., Unix, Windows)
• Network barriers – HTTP and SMTP are standardized network protocols

WS can serve as an enabling technology for application integration. WS, as mentioned earlier, places these following major standards in focus: XML, SOAP, WSDL, UDDI and HTTP.

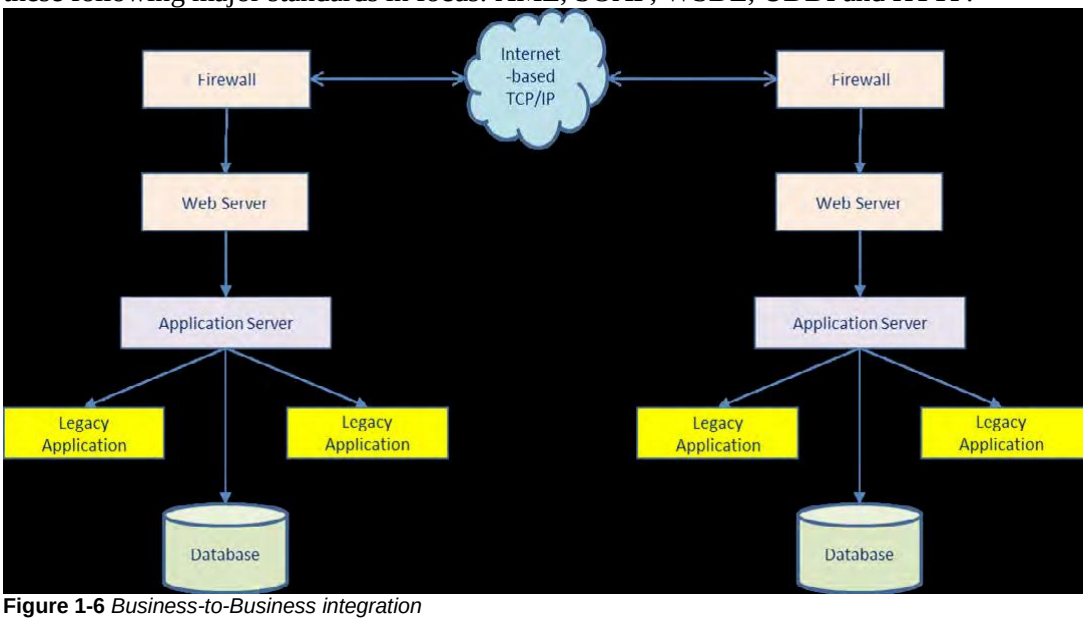

Figure 1-6 *Business-to-Business integration*

SOAP sequence diagram

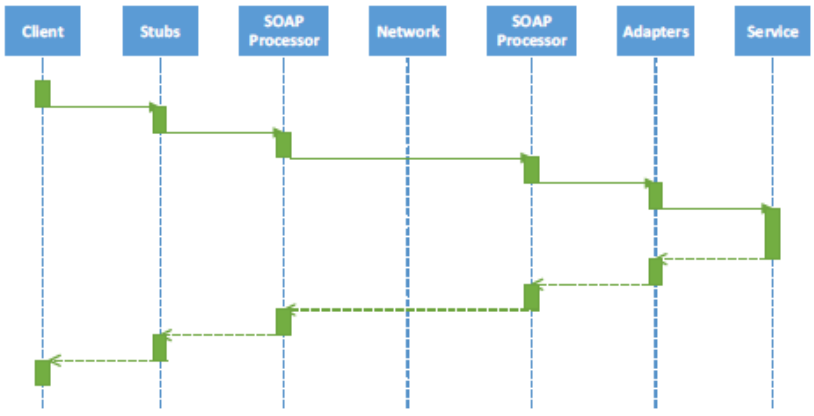

Figure 1-7 *Sequence diagram of SOAP*

Service requester – the client that consumes or requests the service
Service provider – the entity that implements the service and fulfill the service requests

Service registry – a listing like a phonebook where available services are listed and described in full

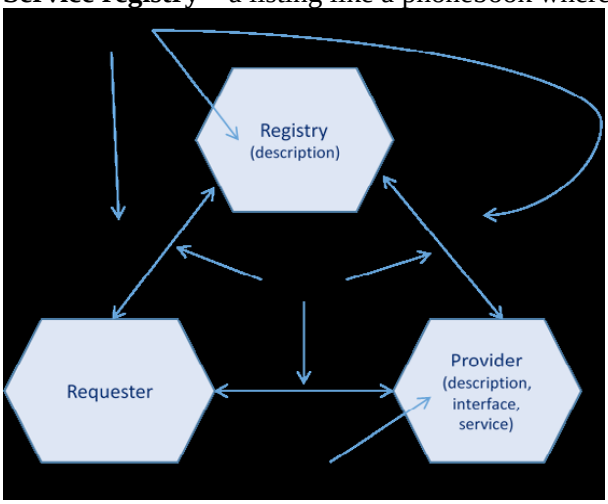

Figure 1-8 *Web Service Architecture*

1.3 Benefits of Web Services

Web Services provide many benefits:

1. Platform-independent: Web Services are now available in nearly all platforms:
a) Hardware: mainframe, midrange, personal and mobile devices
b) Operating systems: UNIX, Windows, Mainframe OS, Android, and iPhones
2. Reuse of existing networking infrastructure: HTTP, SMTP, and JMS protocols
3. Loose-coupling of software components promotes software reuse
4. Reduced integration cost and increased integration speed
5. Open architecture and communication protocols