Overview of SOAP

By Tom Clements
January 2002 (originally posted August 2001)

In the movie *Fight Club*, Brad Pitt and Edward Norton play alter egos -- opposite ends of the psychological spectrum -- two guys trying to communicate with one another and having a tough time making it work. Interestingly enough -- and without giving away the punch line -- much of the action in the film revolves around the production of soap, an activity that seems to bind the characters in unique and unexpected ways.

Now fast-forward to a scenario of a different kind, one played out on the Internet between two software alter egos, Microsoft and Sun, each with a well-defined point of view, each trying to bridge the gap and establish lines of communication with the other. Enter SOAP. Simple Object Access Protocol.

**Introduction**

SOAP, to put it simply, allows Java™ objects and COM objects to talk to each other in a distributed, decentralized, Web-based environment.

More generally, SOAP allows objects (or code) of any kind -- on any platform, in any language -- to cross-communicate. At present, SOAP has been implemented in over 60 languages on over 20 platforms. Suddenly objects everywhere, local and remote, large and small, are able to interoperate. Brad Pitt and Edward Norton, like two very different object types, are finally able to communicate.

In this technology review, I will present SOAP initially in the larger context of Web services, as a protocol that is paired with UDDI (Universal Description, Discovery and Integration) to provide registry and messaging services among businesses. I will also discuss the Web-based underpinnings of the emerging publish-find-bind paradigm, and reveal the mechanisms of SOAP packaging, transport and delivery.

**Evolution of Web Services**

Despite all the hoopla, SOAP is simply one component -- albeit a central one --
in the emerging picture of the Web as a standards-based, language- and platform-neutral framework for business operations. These operations are commonly lumped under the generic tag "Web services," but Web services themselves are only as good as the infrastructure that supports them. Accordingly, here's a quick, \( n \)-tiered look at the substrata of the Internet.

**Network Tiers**

Three network tiers are evident in the evolution of Web services: TCP/IP, HTTP/HTML, and XML. These tiers build successively on top of each other and remain compatible today.

The first tier, the TCP/IP protocol, is concerned primarily with passing data across the wire in packets. A protocol that guarantees transmission across public networks, TCP/IP emphasizes reliability of data transport and physical connectivity. Originally the putty holding proprietary networks together, it's now the backbone protocol of the Web on which higher-level, standard protocols such as HTTP rely.

The second tier, HTML over HTTP, is a presentation tier and concerns itself with browser-based search, retrieval and sharing of information. The emphasis here is on GUI-based navigation and the manipulation of presentation formats. In many ways, HTML is more show than go and lacks both extensibility and true programming power. Nonetheless, sharing hypertext-linked documents in a browser-based environment revolutionized the way humans communicate text-based information to one another. Networked desktop environments, burdened with proprietary operating systems and platform dependent software, are slowly but surely giving way to the standards-based, open-systems computing of the Internet.

Leading the charge into this brave new standards-based world is XML, the third and possibly the most compelling tier on the Internet. XML, a strongly-typed data interchange format, provides a new dimension to the HTTP/HTML tier, one in which machine-to-machine communication is made possible through standard interfaces. This layer -- variously described as A2A (application to application), B2B (business to business) or C2C (computer to computer) -- allows programs to exchange data formatted in a platform- and presentation-independent manner. XSLT style sheets may be added as an optional presentation and/or transformational component.

**XML: The Key to Describing Web Services**

The key to making all this possible is machine-to-machine communication, an area in which XML excels. As a syntax for describing data, XML is definition-driven (through the use of DTDs and schemas) and allows information to be manipulated programmatically. This means that most of the guess work can be taken out of B2B communication. Tags can be agreed upon, interfaces can be defined and processing can be standardized. Web services are reusable component programs that utilize XML as a standard, extensible communication framework to facilitate this type of computer-to-computer communication.
Web services provide interfaces for the transport of component data and business logic across HTTP. A huge amount of data sits right behind server-side scripts and in legacy repositories, waiting to be accessed by Web browsers or client applications. Web services promise to revitalize corporate software assets now lying dormant in many enterprise domains.

XML plays a crucial role in integrating Web-resident data into enterprise applications and coordinating the business logic that holds component pieces together. Specific business tasks and services (including workflow logic, business logic, component sequencing logic, transaction logic and so on) can be encapsulated in XML documents and integrated into existing business environments. This allows businesses to leverage existing internal assets and processes and expose this information as Web services, facilitating business transactions and supply chain interaction across the Web.

Because XML is human-readable and text-based, it is ideal as a transport framework for loosely coupled Web services. The bottom line is this: automated transactions increase productivity, reduce costs and improve services. The presence of standards on the net make automated transactions possible, resulting in productivity gains across the board.

SOAP is a technology that derives from an earlier XML-based standard (XML-RPC) and, in some sense, points toward an emerging standard called ebXML (electronic business XML). ebXML is a work-in-progress, geared toward providing a comprehensive definition of shared business messages among trading partners. SOAP is more modest in scope and less complex in implementation.

Loosely Coupled Systems

Web services decouple objects from the platforms that hold them hostage. That is, Web services facilitate interactions among platform-independent objects, which are able to access data from anywhere on the Web. As part of the movement away from proprietary platforms, Web services rely on loose, rather than tight, couplings among Web components. According to Brian Travis, SOAP consultant and author, "Systems that rely on propriety objects are called tightly coupled because they rely on a well-defined but fragile interface. If any part of the communication between application and service object is disrupted, or if the call is not exactly right, unpredictable results may occur." EDI is an example of a tightly-coupled framework for doing electronic commerce. Loosely coupled systems allow for flexible and dynamic interchange in open, distributed Web environments.

Second Coming of CORBA

Companies on the net today -- IBM, BEA, Sun, to name a few -- simultaneously cooperate with the very companies they compete against. Standardized network transport protocols, platform-independent programming languages like Java, XML and industry-specific dialects, and open component-based server architectures all contribute to this non-proprietary free-for-all. Now Web
services, with its promise of broad-based application interoperability, arrives on the scene as the ultimate "glue" to make these technologies interact, if not seamlessly, at least without the excess baggage that accompanied previous technologies like CORBA and RMI.

In some sense, Web services represent the second coming of CORBA. But whereas CORBA was an object-oriented, IIOP-based binary communications framework, laden with stubs, skeletons and vendor-specific ORBs, Web services are lightweight, HTTP-based, XML-driven, and completely platform- and language-neutral. Where CORBA was a 600-pound semi-caged gorilla, Web services is a gazelle romping freely through the vast Internet preserves.

**Publish, Bind, and Find**

A Web services framework consists of a publish-find-bind cycle, whereby service providers make data, content or services available to registered service requesters who consume resources by locating and binding to services. Requesting applications tune themselves to Web services using WSDL (Web Services Description Language), which provides low-level technical information about the service desired, grants applications access to XML Schema information for data encoding, and ensures that the right operations are invoked over the right protocols.

Publish, bind and find mechanisms have their respective counterparts in three separate (but somewhat equal) protocols that make up the Web services network stack: WSDL, SOAP and UDDI (Universal Description and Discovery Interface).

To drill down a little deeper on the CORBA analogy, SOAP plays the role of IIOP in CORBA (or JRMP in RMI). It's the binding mechanism between conversing endpoints. WSDL, on the other hand, plays the role of IDL (Interface Definition Language). In this capacity, WSDL defines Web services as a collection of ports and operations. A WSDL port is analogous to an interface, and a WSDL operation is analogous to a method. WSDL publishes Web service interfaces to parties interested in communicating across heterogeneous platforms.

WSDL, however, goes beyond just being an interface definition language; it also includes constructs that let you describe address and protocol information for the Web services you want to publish. The interesting thing about WSDL is that it describes an abstract interface for Web services while simultaneously allowing you -- in excruciating detail -- to bind a Web service to a specific transport mechanism, such as HTTP. By abstracting the interface, WSDL functions as a reusable Web service technology. By binding to a specific transport mechanism, WSDL makes the abstract concrete. If this seems somewhat schizophrenic, just think of the space shuttle: a reusable, fully functioning space capsule bound to dedicated, but expendable, booster rockets. The transport mechanism may change, but the payload persists.

Finally, UDDI acts as a registry for publishing and locating Web services. By exposing service information and binding interfaces in a Web-based registry,
UDDI provides a shared directory for businesses and customers to locate one another's Web services.

**Building Web Services**

SOAP lets you build applications by remotely invoking methods on objects. SOAP removes the requirement that two systems must run on the same platform or be written in the same programming language. Instead of invoking methods through a proprietary binary protocol, a SOAP package uses XML, a text-based syntax for making method calls. All information between the requesting application and the receiving object is sent as tagged data in an XML stream over HTTP. From a Web services point of view, SOAP may be implemented as either a client or a server.

**SOAP Clients and Servers**

A SOAP client is a program that creates an XML document containing the information needed to invoke remotely a method in a distributed system. SOAP clients need not be traditional. In addition to being your garden-variety desktop application, a SOAP client could also be a Web server or a server-based application.

Messages and requests from SOAP clients are typically sent over HTTP. As a result, SOAP documents are able to traverse almost any firewall, enabling the exchange of information across divergent platforms.

A SOAP server is simply special code that listens for SOAP messages and acts as a distributor and interpreter of SOAP documents. External Web services may interact with application servers based on J2EE™ technology, which process SOAP requests from a variety of clients.

SOAP servers ensure that documents received over a HTTP connection are converted to a language that the object at the other end understands. Because all communications are made in the form of XML, objects in one language (say, Java) may communicate through SOAP with objects in any other language (C++, for example). It's the job of the SOAP server to make sure the end points understand -- and are happy with -- the SOAP they're being served.

**SOAP and Java Technology**

According to the SOAP 1.1 specification, SOAP is "a lightweight protocol for exchange of information in a decentralized, distributed environment."

SOAP does not mandate a single programming model -- nor does it define language bindings for a specific programming language. In the context of the Java programming language, it's up to the Java community to define the specific language binding. Java language bindings are now being pursued through the JAX-RPC initiative.
In a recent talk on SOAP given at the JavaOne℠ developer conference, Sun engineers Roberto Chinnici and Rahul Sharma defined the role of the JAX family of technologies as "enabling the creation of Web services using the familiar JSP™ and EJB™ component technologies for the Java platform." Servlets and stateless session beans were cited as the two most likely Java technologies for encapsulating Web services.

**So What Is SOAP, Really?**

Having thoroughly set the stage for SOAP and described its crucial role in Web services, let's take a closer look at what SOAP really is, what it does, and how it does it.

SOAP is an extensible, text-based framework for enabling communication between diverse parties -- in general, objects -- that have no prior knowledge of each other or of each other's platforms. From the point of view of objects on the net, SOAP is the ultimate blind date. Client applications can interoperate in loosely-coupled environments to discover and connect dynamically to services without any previous agreements having been established between them.

SOAP is extensible, because SOAP clients, servers and the protocol itself can evolve without breaking existing apps. SOAP, moreover, is generous in terms of supporting intermediaries and layered architectures. This means processing nodes can sit on the path a request takes between the client and server. These intermediate nodes process parts of the message specified by SOAP through the use of headers, which allow clients to identify which node works on what part of the message. This type of intermediate header processing is performed by private contract between the client application and the intermediate processing node. SOAP provides a `mustUnderstand` attribute for headers, which allows the client to specify whether the processing is mandatory or optional. If `mustUnderstand` is set to 1, the server must either perform the intermediate processing specified in the header or throw a fault.

SOAP also defines data encoding rules, called base level encodings or "Section 5" encodings, from the section of the SOAP spec that describes them. It should be pointed out that SOAP encodings take up most of the space in the 40-page SOAP 1.1 spec. Without getting bogged down too deeply in XML data type specifics -- a swamp still being drained by experts over at the XML Schema group -- SOAP encodings can be described briefly as a collection of either simple or compound values.

Simple values are either simple types, like ints, floats and strings, or built-in types as defined in Part 2 of the XML Schema specification. These include data types such as arrays of bytes and enumerations.

Compound values include structures, arrays and complex types as defined -- again -- by the XML Schema group. Last, but certainly not least, SOAP data encodings specify rules for object serialization; that is, mechanisms for marshaling and unmarshaling data streams across the net. It's important to
note that these "Section 5" encodings are not mandatory in any way, so clients and servers are free to use different conventions for encoding data as long as they agree on format. Doing so, however, countermands the push SOAP gives to standardized services on the net and comes with the usual caveat: your mileage may vary. Individual clients and servers are nonetheless free to take the road less traveled.

Finally, SOAP establishes a set of rules that enable clients and servers to do remote procedure invocation using SOAP as a communications framework. SOAP -- at base a message-oriented protocol -- can, with these conventions, work well as an RPC-type protocol. Object serialization is the mechanism that gives SOAP-RPC its punch.

Message Format

SOAP does all this in the context of a standardized message format. The primary part of this message has a MIME type of "text/xml" and contains the SOAP envelope. This envelope is an XML document. The envelope contains a header (optional) and a body (mandatory). The body part of the envelope is always intended for the final recipient of the message, while the header entries may target the nodes that perform intermediate processing. Attachments, binary or otherwise, may be appended to the body.

SOAP provides a way for the client to specify which of the intermediate processing nodes has to deal with what header entry. Because headers are orthogonal to the main content of the SOAP message, they're useful in adding information to the message that doesn't effect the processing of the message body.

Headers, for example, may be used to provide digital signatures for a request contained in the body. In this circumstance, an authentication or authorization server could process the header entry -- independent of the body -- stripping out information to validate the signature. Once validated, the rest of the envelope would be passed on to the SOAP server, which would process the body of the message. A closer look at the SOAP envelope will help to clarify the placement and purpose of SOAP header and body elements.

Anatomy of a SOAP Envelope

The SOAP 1.1 spec provides the following sample envelope:

```xml
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Header>
   <t:Transaction xmlns:t="some-URI">
     SOAP-ENV:mustUnderstand="1"
   </t:Transaction>
 </SOAP-ENV:Header>
</SOAP-ENV:Envelope>
```
In this example, a GetLastTradePrice request is being sent to a stock-quote service somewhere on the Web. The request takes a string parameter, a ticker symbol, and returns a float in the SOAP response.

The SOAP envelope is the top element of the XML document that represents the SOAP message. XML namespaces are used to disambiguate SOAP identifiers from application specific identifiers. XML namespaces are used heavily in SOAP to qualify or scope elements in the message to a specific domain. To understand SOAP namespaces, it helps to be familiar with the namespace spec for XML. If you're not, simply think of namespaces as neighborhood identifiers that help uniquely identify SOAP elements by associating them with specific locations, real or imagined.

**Namespaces**
The first namespace in the example references the SOAP schema, which defines the elements and attributes in the SOAP message. The second namespace refers to SOAP encodings, the "Section 5" data types discussed earlier. Since no additional per-element encoding is specified, this encoding applies to the whole document.

**Header**
The first element identified in this sample SOAP envelope header is a transaction element, accompanied by a namespace attribute and by the `mustUnderstand` attribute with a value of 1. Since `mustUnderstand` is set to 1, the server accepting this message must perform intermediate processing on this transaction node. You can interpret this to mean that the server and client have previously agreed upon the semantics that govern the processing of this header element, so that the server knows exactly what to do with the contents of the element, in this case 5.

If the server receiving this message doesn't understand the semantics of the transaction header, it is required to reject the request completely and throw a fault. A fault element is a special part of the SOAP body and a well-defined mechanism to ship error information back to the client.

Intermediate processing nodes like this are an example of SOAP's extensibility. Clients include such nodes in a SOAP message to indicate that special processing needs to take place before the contents of the message body can be processed. Ensuring backward compatibility with existing servers not capable of providing such processing is simply a matter of setting the `mustUnderstand` attribute to 0, which makes the action optional.

In addition to defining transaction nodes like the one described above, a SOAP
message may optionally contain header entries specifying nodes that perform authorization processing, encryption, persistence of state, business logic processing and so on. Headers help make SOAP a modular, extensible packaging model. Just keep in mind that header processing is entirely independent of the SOAP message body.

Body
The SOAP body in the example contains an XML payload, which we can surmise, without really seeing it spelled out for us, does RPC. SOAP is not only a modular packaging model, it's also a fairly cryptic packaging model.

Nothing here explicitly shows that RPC is begin done. All we see in the body are a couple of XML elements, one qualified by a namespace. It's up to the SOAP server to understand the document semantics and do the right thing. The server, in effect, provides a framework for dealing with the XML payload in a meaningful way. "Meaningful" here implies that the server invokes a remote procedure call on some back-end database to receive the stock price for the stock-symbol element contained in the message body. All the magic takes place behind the SOAP RPC curtain.

SOAP-RPC

SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but SOAP messages are often combined to implement request/response mechanisms. To do RPC using SOAP, a few conventions must be followed. First of all, request and response messages must be encoded as structures. For each input parameter of an operation, there must be an element (or member of the input structure) with the same name as the parameter. And for every output parameter, there must be an element (or member of the output structure) with a matching name.

Here's a foreshortened, RPC-based view of the SOAP message presented earlier. Only the body portions of the SOAP request and response envelopes are shown.

Request
<SOAP-ENV:Body>
  <m:GetLastTradePrice xmlns:m="some-URI">
    <symbol>DEF</symbol>
  </m:GetLastTradePrice>
</SOAP-ENV:Body>

Response
<SOAP-ENV:Body>
  <m:GetLastTradePriceResponse xmlns:m="some-URI">
    <price>22.50</price>
  </m:GetLastTradePriceResponse>
</SOAP-ENV:Body>

The request invokes the GetLastTradePrice method. Notice the response defines a GetLastTradePriceResponse operation. A convention common to
SOAP calls for appending `Response` to the end of a `Request` operation to create a `Response` structure. This output structure contains an element called `price`, which returns the results of the method invocation, presumably as a float.

It's important to note that nowhere in the SOAP envelope are data types explicitly delineated, so we really don't know the type of the symbol or the type of the result parameter `price` just by looking at the SOAP message. Client applications define data types either generically through "Section 5" encodings, or privately via agreed-upon contracts with servers. In either case, these definitions are not explicitly included in the SOAP message.

Finally, in order to do RPC, a lower-level protocol like HTTP is needed. Although the SOAP 1.0 specification mandated the use of HTTP as the transport protocol, SOAP 1.1 (and its sister specification "SOAP Message with Attachments") permit the use of FTP, SMTP or even (possibly) raw TCP/IP sockets. All the serialization and encoding rules general to SOAP apply to RPC parameters as well.

**A SOAP Use Case**

![Figure 1: Processing steps in a SOAP use case](Click image to enlarge.)

Now that you've seen the SOAP envelope in detail, it helps to pull back a little and view SOAP from a use-case perspective to grasp the round-trip processing that takes place in a distributed Web environment. Here are a few broad, summary propositions that form the conceptual backbone of Web services and SOAP.

- A client application somewhere on the Internet consumes Web services.
- Web services (via SOAP) expose object methods.
- Object methods access remote data anywhere on the Web.

Applying transitive logic to these network propositions, we can deduce the whole point of Web services and SOAP: Clients somewhere consume data anywhere on the Web.

QED.

Here's the use case in a little more detail.

1. A SOAP client uses the UDDI registry to locate a Web service. Rather than manipulate WSDL directly, in most cases a SOAP application will be hardwired to use a particular type of port and style of binding, and it will dynamically configure the address of the service to be invoked to match the ones discovered through UDDI.

2. The client application builds a SOAP message, which is an XML document capable of performing the desired request/response operation.
3. The client sends the SOAP message to a JSP or ASP page on a Web server listening for SOAP requests.

4. The SOAP server parses the SOAP package and invokes the appropriate method of the object in its domain, passing in the parameters included in the SOAP document. Optionally, intermediate processing nodes may have performed special functions as indicated by SOAP headers prior to receipt of the message by the SOAP server.

5. The request object performs the indicated function and returns data to the SOAP server, which packages the response in a SOAP envelope. The server wraps the SOAP envelope in a response object, such as a servlet or a COM object, which is sent back to the requesting machine.

6. The client receives the object, strips off the SOAP envelope and sends the response document to the program originally requesting it, completing the request/response cycle.

**Summary**

SOAP is an XML-based protocol for sending messages and making remote procedure calls in a distributed environment. Using SOAP, data can be serialized without regard to any specific transport protocol, although HTTP is typically the protocol of choice.

SOAP is good for building platform and language-neutral systems that interoperate. Overall, SOAP and Web services account for everything needed to build a distributed application infrastructure on top of XML. SOAP minimizes the problem of multiple-platform incompatibilities in accessing data by resolving the conflict between the COM and Java component object models.

To return to the metaphor that kicked off this discussion, SOAP is the perfect medium for object entities of all types -- even Hollywood types like Brad Pitt and Edward Norton -- to use as a communications medium. As in the film, expect the results of this new technology to be earth-shaking.

**References**


*Understanding SOAP*, Kennard Scribner and Marc C. Stiver, Sams Publishing, 2000. (This is a fascinating, highly technical exposition of SOAP and network technologies by two guys who have "been there, done that." Extremely low-level detail, presented in a fluid style. A great book for developers experienced
in XML and Web services looking to be experts in SOAP.)


XML and SOAP Programming for BizTalk Servers, Brian E. Travis, Microsoft Press, 2000. (Don't let the title throw you off. This is an excellent, high-to-medium level view of Web services and SOAP, brilliantly written and extremely well-organized. BizTalk takes a back seat to the more generic preliminaries that make up the greater portion of the text. An absolutely great book for beginners.)

About the Author

Tom Clements is a freelance tech writer and poet, specializing in Java™ API documentation, XML/XSLT, device-driver writing and wireless communications.

Reader Feedback

Tell us what you think of this article.

- Very worth reading
- Worth reading
- Not worth reading

If you have other comments or ideas for future articles, please type them here:

Submit

Have a question about programming? Use Java Online Support.
Important characteristics of SOAP SOAP protocol is based on XML and hence platform independent. Though HTTP is the most commonly used transport. SOAP based webservices uses a special form of XML called SOAP to exchange messages (requests and responses). SOAP originally stood for Simple Object Access Protocol, but now also called as Service Oriented Architecture (SOA) protocol. Important characteristics of SOAP. SOAP protocol is based on XML and hence platform independent. SOAP is known as the Simple Object Access Protocol, but in later times was just shortened to SOAP v1.2. SOAP is a protocol or in other words is a definition of how web services talk to each other or talk to client applications that invoke them. SOAP was developed as an intermediate language so that applications built on various programming languages could talk easily to each other and avoid the extreme development effort. In this SOAP Web services tutorial, you will learn:- SOAP Introduction. Advantages of SOAP. Soap is an excellent cleanser because of its ability to act as an emulsifying agent. An emulsifier is capable of dispersing one liquid into another immiscible liquid. This means that while oil (which attracts dirt) doesn't naturally mix with water, soap can suspend oil/dirt in such a way that it can be removed. The organic part of natural soap is a negatively-charged, polar molecule. Its hydrophilic (water-loving) carboxylate group (-CO2) interacts with water molecules via ion-dipole interactions and hydrogen bonding.