Making SOAP with Soup

Alex Graveley
Ximian Inc.
alex@ximian.com, http://www.ximian.com

Abstract

In this paper we explore existing remote procedure call mechanisms and present an alternative more suited to Internet transactions. This mechanism, Simple Object Access Protocol (SOAP), uses XML for data and type encoding and is transferred over protocols such as HTTP and SMTP. An implementation of SOAP written in C, named Soup, will be discussed in detail.

Soup is designed to be fully asyncronous, and provides skeleton code generation for use with SOAP clients and servers. WSDL, a SOAP interface description language is covered as well as future SOAP developments such as object discovery and integration between existing RPC mechanisms.

1 Introduction

Programmatic use of remote computing resources has become a common paradigm in enterprise level application development. Since their initial adoption as viable systems, numerous incarnations of so-called Remote Procedure Call (RPC) mechanisms have been developed.

Each of these systems exhibit distinguishing features which determine their best usage scenarios. Wire encoding, data type capabilities, object system, speed, active deployment, and initial learning curve all play important roles when choosing the correct RPC for a given application. However, many of these existing systems are designed for use in strongly controlled environments, such as LANs or extranets. This leads to limited client accessibility in highly diverse Internet transactions. In these cases, restrictions like unstable connections or strict firewall setups can make access by certain clients nearly impossible.

Simple Object Access Protocol (SOAP) is a recently standardized specification for an RPC system which attempts to solve this problem. Based strongly around existing Internet standards, SOAP uses transport protocols like HTTP or SMTP for communication between client and server, and XML for on the wire data encoding.

This paper discusses the design goals and protocol internals of the SOAP specification, as well as the XML Schema-based type system used by SOAP, and the service description language used to define SOAP interfaces. We also introduce a SOAP implementation with C language bindings named Soup.

2 SOAP Design Goals

SOAP is designed to be simple, leveraging existing standards where possible and, building upon a strong extensible XML base. The simplicity of the client/server protocol and the strength of the type system allow it to cover a wide range of capabilities while keeping the specification clean.

Transport Agnostic

> The specification does not specify which transport protocols can be used to carry SOAP messages. Instead it defines a means by which structured data can be encapsulated generically in existing protocols. Standardization on how best to encapsulate SOAP in other protocols is left to outside efforts.

Object System Agnostic

> The specification does not specify any sort of object activation, discovery, reference counting,

or life cycle management. This allows for simplest case transactions to occur with a minimum of programmer overhead, while also allowing complex systems to be supported.

Language Agnostic

The specification does not define any language bindings. This allows SOAP messages to be constructed in whichever way is most suitable for a given programming environment and underlying transport protocol. In fact, SOAP implementations for many common languages such as C, C++, Visual Basic, Perl, and Python already exist.

Internet Friendly

In practice, SOAP works well through firewalls, with unreliable connections, is easily encryptable, and easy to debug and trace, all of which make it exceptional for use in Internet situations. In order to function well in this evironment, SOAP avoids specifying features like server to client callbacks, or TCP/IP port schemes, unlike several other RPC mechanisms which make heavy use of them.

SOAP is actually divergent from most of the newer RPC mechanisms, holding simplicity and extensibility as paramount. In fact, SOAP abstains from many common RPC idioms which would limit its extensibility and possible uses. This leaves it in the unique position to allow bridging between other RPC mechanisms.

Already SOAP has been used to allow communication with the Microsoft COM/DCOM component system, and work is under way to specify a CORBA/SOAP bridge.

3 Transport Encapsulation

Data is tranferred between client and server using SOAP messages, which are simply structured XML payloads. A client request message is composed of a server-unique object locator, the SOAP method name to be invoked of the object identified, and method parameter data. A server response message encodes return result data or optionally a SOAP fault used to signal some error condition back to the requesting client. Both client and server messages may contain arbitrary SOAP headers, which are out-of-band data for processing by the destination (not to be confused with the headers used by many transport protocols).

Locating a specific object on which to invoke a SOAP method, and the means to do so is left solely to the transfer protocol implementation. In the case of HTTP transport, a SOAPAction HTTP header must be present in order to identify the resource being requested. This header will contain an arbitrary URI. In the case of SMTP the mail address of the recipient will be used to identify the message processor. The SOAP specification itself does not describe how this information is originally acquired, but leaves it up to external mechanisms such as a Service Description Language, explained later.

Request parameters and response data are encoded as a child element of a `Body` XML element. This `Body` element is a direct child of the message's top-level `Envelope` element. SOAP headers are transferred inside a `Header` element also a direct child of the `Envelope` element.

```
<SOAP-ENV:Envelope>
    <SOAP-ENV:Header>
        <session-id value='b4f541609c844180'/>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <GetCurrentTemperature>
            <city>Boston</city>
            <country>US</country>
        </GetCurrentTemperature>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 1: Simple SOAP request message

Figure 1 shows a sample SOAP request message body. This message body would remain the same were is being delivered using HTTP, SMTP, or any other transport protocol. The `session-id` element is a SOAP header. The `GetCurrentTemperature` element identifies the SOAP method to be invoked upon the destination object. The `city` and `country` elements are method parameters.

```
<SOAP-ENV:Envelope>
```

```
    <SOAP-ENV:Body>
        <GetCurrentTemperatureResult>
            <temperature units='celsius'>
                15
            </temperature>
        </GetCurrentTemperatureResult>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2: Simple SOAP response message

Figure 2 shows a possible SOAP response to the request issued in Figure 1. The `GetCurrentTemperatureResult` element marks that this message is a response to a `GetCurrentTemperature` method invocation, and the `temperature` element is the returned value.

Work on standardizing how SOAP is to be encapsulated inside other transport protocols is under way. Currently HTTP and SMTP transfer is standardized, and the DIME protocol which allows for SOAP messages tranferred directly over TCP or UDP is under development.

4 XML Schema Type System

XML Schema is the recently finalized system which replaces DTD as the mechanism for validation and description of an XML document. It supports many more features than did DTD, and is defined using structured XML versus DTD's element-based system. SOAP uses XML Schema as its type system for datatype definition, and incoming XML document validation.

Schema allows common concepts like basic types, lists, unions, and complex types which are a collection of basic types or other complex types. In addition, schema allows new types to be derived from other types in highly extensible ways.

4.1 Simple Types

Schema's set of basic types is quite large. It includes strings, integers, bytes, floating point numbers, dates, time durations, binary data encoded as Base-64 or hex, and URIs, to name a few. There are also variations to support for example only a positive integer, or an unsigned byte.

Derivation from simple types allows restrictions to be imposed on instances of the new type. Instance values in an XML document must comply with all restrictions in order to be considered valid XML. The basic set of these restrictions (called facets) allow you to do things like limit the length of strings, or force them to conform to a regular expression pattern, set arbitrary upper or lower bounds on numeric values, limit their number of integral or fractional digits, and explicitly enumerate allowed values for an instance.

```
<simpleType name='temperatureUnits'>
    <restriction base='xsd:string'>
        <enumeration value='celsius'/>
        <enumeration value='fahrenheit'/>
        <enumeration value='kelvin'/>
    </restriction>
</simpleType>
```

Figure 3: `temperatureUnits` *simple type definition*

Figure 3 defines the simple type `temperatureUnits`. `temperatureUnits` is derived from the base type `string`, and imposes a `restriction` on the allowed string values using an enumeration. This means that any instace where the `units` attribute appears, must have a value of "celsius", "fahrenheit", or "kelvin".

4.2 Attributes

Attributes are a common XML idiom. They are name/value parameters which may appear in a complex type element. In Schema, attributes can be specified explicitly, and given a simple type which the attribute value must conform to. Various aspects of the attribute can also be specified, such as default value, or whether its appearance in an instance is mandatory.

```
<attribute
    name='units'
    type='temperatureUnits'
    use='required'/>
```

Figure 4: `units` *attribute definition*

Figure 4 defines the attribute `units` used in Figure 2. The `name` attribute of the definition identifies how this created attribute will be named in an

XML instance element. The `type` attribute tells us that instances of this attribute will be of the `temperatureUnits` type (defined in Figure 3), and hence must be one of the explicit values allowed for that type. Finally, the `use` attribute says that complex type definitions which make use of the `units` attribute will always have this attribute present in instances.

Schema also has the concept of an attribute group, which is a named group of attributes that can be used in more than one type to assure they all share a common set of attributes.

4.3 Complex Types

Complex types are constructed by either listing the simple or complex type member elements it is composed of, or by restricting or extending an existing simple type or complex type.

When restricting a simple type, we are allowed to add additional facets to the base type, or to add element attributes which may appear in an instance document. When extending a basic type, we are only allowed to add attributes.

When restricting an existing complex type, you may add elements, modify existing elements, add attributes, and modify existing attributes. When extending a complex type you may only add elements and attributes. Use of extension is very much like sub-classing the base complex type, to use a common programming term.

When defining the elements of a complex type, there are a variety of options for how they should be represented in an XML instance of the type. You may have `all` elements required (much like a structure in the programming world), a single `choice` of the elements (much like a union), or a `sequence` of the listed elements, any of which may be defined as mandatory or optional. These contained elements may also have a maximum and minimum occurance count, allowing more than one to appear in a give instance. These element groupings can also be chained and nested, to allow very complex sequences of data where necessary.

```
<complexType name='temperatureType'>
    <simpleContent>
        <extension base='xsd:decimal'/>
            <attribute ref='units' use='required'/>
        </extension>
    </simpleContent>
</complexType>
```

Figure 5: `temperatureType` *complex type definition*

Figure 5 defines a new complex type named `temperatureType` which is used as the type for the `temperature` element in Figure 2. `temperatureType` uses extension to sub-class the decimal simple type. It also adds a new attribute which references the attribute we created in Figure 4. The `use` attribute of this new attribute is set to "required" which means that the `units` attribute must be present in all instances of a `temperatureType`.

4.4 Elements

Elements are required to differentiate between simple and complex type definitions which are meant only for internal use, for example an abstract base class, and those which are allowed to appear in an XML instance document conforming to the schema specified. It also binds instance element names to a type definition.

```
<element name='city' type='xsd:string'/>
<element name='country' type='xsd:string'/>

<element
    name='temperature'
    type='temperatureType'/>
```

Figure 6: Example element definitions

Figure 6 defines the elements `city`, `state`, and `temperature` which are used in Figures 1 and 2. The `city` and `state` elements are bound to the Schema `string` type, as no other specialization is needed. The `temperature` element is bound to the `temperatureType` complex type definition we created in Figure 5.

5 Web Services Description Language

Web Services Description Language (WSDL) is a corollary to an Interface Definition Language (IDL) used by other RPC mechanisms such as CORBA, COM, and Sun RPC. WSDL identifies the methods and their parameters available to be called at a given URL. The destination URL identifies and defines a so-called "web service", which represents a set of methods with a common association, somewhat like a `class` in object-oriented programming. By using a destination URL WSDL also describes the transport to be used when communicated with a given service. Keeping this transport to SOAP method mapping contained a WSDL description allows programmers to remain ignorant of the underlying transport mechanism being used.

```
<definitions
    name="Thermometer"
    targetNamespace=
"http://ximian.com/temp.wsdl"
    xmlns:temp=
"http://ximian.com/temp.xsd"
    xmlns:soap=
"http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl=
"http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

<message name="GetCurrentTemperatureInput">
    <part
        name="city"
        element="temp:city"/>
    <part
        name="country"
        element="temp:country"/>
</message>
<message name="GetCurrentTemperatureOutput">
    <part
        name="temperature"
        element="temp:temperature"/>
</message>

<portType name="TemperaturePortType">
    <operation name="GetCurrentTemperature">
        <input message=
"GetCurrentTemperatureInput"/>
        <output message=
"GetCurrentTemperatureOutput"/>
    </operation>
</portType>

<binding
    name="TemperatureBinding"
    type="TemperaturePortType">
    <soap:binding
        style="document"
        transport=
"http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetCurrentTemperature">
        <soap:operation
            soapAction=
"http://ximian.com/temp"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>

<service name="TemperatureService">
    <port
        name="TemperaturePortType"
        binding="TemperatureBinding">
        <soap:address
            location="http://ximian.com/temp"/>
    </port>
</service>

</definitions>
```

Figure 7: Example WSDL definition

Figure 7 is an example WSDL definition of the SOAP messages presented in Figures 1 and 2, which uses the type information specified in Figures 3, 4, 5, and 6 to map types to SOAP method parameters and return values. It also maps the service to the URL "http://ximian.com/temp", which tells us that this service uses an HTTP transfer protocol. Based on the combined information in the WSDL and XML Schema definitions, a WSDL compiler can generate programming language declarations that allow the service described to me called and the results manipulated programmatically.

6 Soup

Soup is a SOAP (Simple Object Access Protocol) implementation for the C programming language. It provides an queued asynchronous callback-based mechanism for sending and servicing SOAP requests, and a WSDL (Web Service Definition Language) to C compiler which generates client stubs and server skeletons for easily calling and imple-

menting SOAP methods.

It uses the Glib main loop and is designed to work well with GTK+ applications. This enables GNOME applications to access SOAP servers on the network in an asynchronous fashion (a synchronous operation mode is also supported for those who want it).

6.1 Current Features

- WSDL 1.1 Compiler
- Full HTTP 1.1 transport engine
- SSL support for secure SOAP communication
- HTTP digest authentication support
- Connection cache
- Standalone HTTP server and Apache module for serving SOAP messages

6.2 Future Development

- POP/SMTP tranfer engine
- DIME TCP/UDP transfer engine
- Disco object discovery support
- SOAP Routing Protocol support
- CORBA IDL to WSDL Compiler
- Microsoft COM over SOAP integration
- CORBA over SOAP testbed
- GTK+ Object and Bonobo integration

7 Acknowledgments

The author would like to thank those contributors and developers who made this project possible. Namely Daniel Veillard for his excellant XML library, J.P. Rosevear, Rodrigo Moya, and especially Dick Porter for his extensive work on the WSDL compiler and runtime implementation.

Also, a special thanks to the architects of the various XML, SOAP, and WSDL specifications, including members of World Wide Web Consortium and employees of Microsoft Corporation, Sun Inc., IBM Inc., and DevelopMentor.com.

8 Availability

The Soup runtime is available under the LGPL; the WSDL compiler under the GPL. Visit `http://www.gnu.org` for more details.

Download the latest tarball from:
`ftp://ftp.gnome.org/pub/GNOME/unstable/sources/soup`

Or from GNOME CVS, in the `"soup"` module. Visit `http://www.gnome.org/cvs` for more information.

Also, please feel free to join to the Soup discussion list. To subscribe, send mail with the word `Subscribe` in the message body to `soup-list-request@ximian.com`.

9 Online Resources

To learn more about SOAP, XML Schema, or WSDL visit these web sites:

- XML Schema Home
  `http://www.w3.org/XML/Schema`
- SOAP 1.1 Specification
  `http://www.w3.org/TR/SOAP`
- WSDL 1.1 Specification
  `http://msdn.microsoft.com/xml/general/wsdl.asp`
- Developmentor SOAP Home
  `http://www.develop.com/soap/default.htm`
- Microsoft SOAP Home
  `http://msdn.microsoft.com/soap/default.asp`
- Gnome-XML Library
  `http://xmlsoft.org/xml.html`