

Roland's Best Practices [Web Services]

General Guidelines

- Should be obvious, but, solution should be programming language and hardware/software platform agnostic.
- Use SOAP
support both SOAP 1.1 and SOAP 1.2 requests/responses
- Use only document-style SOAP messages. RPC-style seems overly complex and not very interoperable. A document Request/Response model can easily map into a RPC-like behavior.
- Try not to use any SOAP headers -- stick with a SOAP Envelope, and a SOAP Body with one element that is the document.

- Use XML Schema to define all documents. All documents will have namespaces, and the solution should allow for easy sharing of elements between schemas.

i.e., the SUNetID2 WS schema should be able to reference the Person element from the Person schema.

- When interoperability is a concern, support SOAP over HTTP (for unauthenticated web services) and SOAP over HTTPS with client certificates for authenticated web services.
- When interoperability is not a concern (i.e., we can give the clients a jar file with client stubs), use a GSS socket, with the equivalent HTTP headers/soap body being placed inside the encrypted channel. Allow the connection to be left open and re-used when the client wants to make a lot of SOAP calls all at once.
- Note that this also has the potential of being used as a proxy/tunnel with a vanilla HTTP SOAP client. Someone would run the proxy on localhost, and point their SOAP client the proxy, which would connect to the GSS socket on the server.

- Provide jar files that contain the client-side APIs of the SOAP service. The developer using them shouldn't even need to know that they are using SOAP/XML/etc. The client-side stubs should also take care of authentication.
- The “business logic” in the server-side stubs shouldn't really know they are being called via SOAP, or know anything about XML.
- Authentication/Authorization/Logging/Config should be taken care of as much as possible by the server framework.

- Parsing of XML, as well as the binding of XML to native language data structures, should be as efficient as possible.
- Should be easy to develop and deploy new web services
- If performance (XML overhead) becomes a big concern, maintain the ability to use an alternative, more efficient encoding (still using XML schema to specify the documents though).

Implementation Details

- For Java bindings, use the XML Beans framework (originally from BEA, recently part of the Apache XML projects). This framework allows you to convert from XML to Java objects and back again, using XML schema definitions. It uses an XML Pull Parser and appears to be very efficient. It also doesn't require any static configuration files. You just compile bindings into jar files and add them to your class path to support new schemas.
- No need to use SAX or DOM XML apis in any business logic.
- No research has been done on automatically generating language bindings for other languages (like Perl, C), but the task should be straightforward, and hand-crafted bindings are always possible.

- For the SOAP engine, we'll write our own. It will use XML Beans with the standard SOAP 1.2 and 1.1 XML Schemas to ensure we parse and generate valid SOAP messages. The engine will run as servlet (most likely in Tomcat), and support SOAP messages via HTTP, HTTPS and GSSAPI.
- To generate client/server stubs as well as engine configuration, use our own XML config file. Most of the SOAP servers I looked at required both WSDL and other custom config file for things like controlling Java code generation, and client/server deployments. I figure I can keep the configuration extremely simple and have it only contain relevant information. It should also be possible to generate a WSDL 1.1 or WSDL 1.2 from our config file if one is needed.

- The SOAP engine will have support for authentication/ authorization as part of the framework. If we want to support both client SSL certs and GSSPI-API for authentication (might also support the OASIS Kerberos Token profile), then the framework will need to address how clients identities are specified.

