

Internet Computing Project 3

Robert van Engelen

Due date: Dec. 7, 2006

Your task is to implement a composite application based on Web services. We will use gSOAP for this assignment.

To get started:

1. Download the Linux version of gSOAP *gsoap-linux-2.7.9a.tar.gz* from SourceForge:
<http://sourceforge.net/projects/gsoap2>
2. Create a new project directory and unpack the gSOAP distribution package there.
3. The Linux version of gSOAP has pre-built *wSDL2h* and *soapcpp2* executables. Put the *gsoap-linux-2.7.9a/bin/wSDL2h* and *gsoap-linux-2.7.9a/bin/soapcpp2* executables in your *bin* directory or change your *path* environment variable to include *gsoap-linux-2.7.9a/bin*.
4. You will also need the *gsoap-linux-2.7.9a/stdsoap2.h* and *gsoap-linux-2.7.9a/stdsoap2.cpp* files to compile your project.

The composite service application that we will build is a C++ client that invokes Web services. The objective of the client is to list all ski resorts that lie within a certain radius of a given coordinate in the US, ranked according to the favorable weather forecast predictions for the next couple of days. So if we are looking for the best opportunity to enjoy the outdoors, we can make an informed decision.

To build the composite application, we will use two publicly available services: a service that returns a list the ski resorts within a certain radius from a longitude/latitude location, and a weather forecast service that takes a zip code and returns a seven-day forecast for the ZIP code location. The ski resort listing includes zip codes for the US, so we can use the zip code to find out via the weather service if the weather conditions will be favorable for a week at each resort.

1. Download the ski resort service WSDL from:
<http://www.transactionalweb.com/SOAP/globalskilocator.wsdl>
and save as *SkiLocator.wsdl*. Although we can run *wSDL2h* directly on the URL to create the client application, a downloaded copy of the WSDL will be needed for editing because the WSDL has an error (errors are uncommon but do occasionally occur, which means that you need your XML and WSDL knowledge to fix it, see further below). This is a SOAP 1.1 RPC-encoded service.

2. Download the weather forecast service WSDL from:
<http://www.webservicex.net/WeatherForecast.asmx?WSDL>
 and save as *WeatherForecast.wsdl*. This is an SOAP 1.1 document/literal-style Web service.
3. To check the WSDLs for problems, run *wSDL2h SkiLocator.wsdl* and *wSDL2h WeatherForecast.wsdl*. You will notice that errors are generated for *SkiLocator.wsdl*, where certain bindings and messages could not be found. The problem is that one of the namespace prefixes is bound to the wrong namespace URI. This prefix should refer to the WSDL's namespace (and the XML schema namespace in the *wSDL:types*, which uses the same namespace URI). Check the WSDL to identify which prefix has the wrong binding and edit *SkiLocator.wsdl* to correct this problem. The result is that *wSDL2* should run without errors on *SkiLocator.wsdl*. Note: warnings can be ignored.
4. Copy the *typemap.dat* from *gsoap-linux-2.7.9a/wSDL/typemap.dat* and edit this file. Add two new namespace bindings:

```
tw = "http://TRANSACTIONALWEB.COM"
wx = "http://www.webservicex.net"
```

While these are not strictly needed, the prefix bindings declared in *typemap.dat* are used by *wSDL2h* to generate customized code.

5. Run *wSDL2h -t typemap.dat -o proj3.h SkiLocator.wsdl WeatherForecast.wsdl* to generate *proj3.h* for your C++ client. If you are not familiar with STL, use the *wSDL2h -s* flag to eliminate *std::vector* and *std::string*. If you use STL, copy *stlvector.h* from *gsoap-linux-2.7.9a/import* to your project.
6. This is a good opportunity to use *Doxygen* to generate documentation (the *doxygen* executable is installed on the linprog stack). In this case we will use *Doxygen* to extract relevant WSDL service data from the generated *Proj3.h* file. Type *doxygen -g Doxyfile*. Edit *Doxyfile* and specify *PROJECT_NAME=Proj3* and *INPUT=proj3.h*. Run *doxygen Doxyfile* to generate the HTML code for the project. Open *html/index.html* to read the documented *proj3.h* details, such as service operations, parameters, and data types.
7. Run *soapcpp2 -CL proj3.h* to generate the client-side C++ source code *soapStub.h*, *soapH.h*, *soapC.cpp*, and *soapClient.cpp*. You need these for your project, together with *stdsoap2.h* and *stdsoap2.cpp*. Note that *soapcpp2* also generated sample messages that give you an idea what the message exchange will look like.
8. Create a new *proj3.cpp* file and add `#include "locatorSoapBinding.nsmap"` to include all header and XML namespace definitions related to gSOAP and the services. To invoke the ski resort service, try the following code (assuming you are using STL):

```

struct soap *soap = soap_new(); // new context
struct tw__findWithinRadiusResponse r;
if (soap_call_tw__findWithinRadius(soap, NULL, NULL, "35", "-85", "200", r))
    soap_print_fault(soap, stderr);
else if (r._noname)
{ for (int i = 0; i < r._noname->__size; i++)
    cout << "URL=" << r._noname->__ptr[i]->url
        << " ZIP=" << r._noname->__ptr[i]->addresspostalcode << endl;
}
soap_destroy(soap); // delete objects
soap_end(soap); // delete temp data
soap_free(soap); // done and free the context

```

Note that the SOAP encoded array in C/C++ has a size field `__size` and a pointer `__ptr` to an array of pointers to `tw__Entry` objects. The `soap_call_tw__findWithinRadius` function is a stub (proxy). The two NULL arguments indicate that the service's endpoint and SOAP actions are to be used as declared in the WSDL.

9. Compile *proj3.cpp* together with *soapC.cpp*, *soapClient.cpp*, and *stdsoap2.cpp*. Use `g++` option `-DDEBUG` to enable logging of messages in *SENT.log* and *RECV.log* (automatically appended).
10. Complete the program by invoking the weather forecast service for each ZIP code of the resort and sort the resorts in order of most favorable temperatures on average for the upcoming days, where the maximum and minimum temperature deviation is measured from the ideal 28F. You can decide on the best formula. Your program should accept the longitude, latitude, and radius from the command line and display the ranked resorts (with some details). In case a resort does not list a ZIP code, it should be displayed at the bottom with a comment. Note: recall that `soap_destroy` and `soap_end` remove deserialized data, so use only at the end of the session.
11. Submit your program source codes, Makefiles, etc to me in a tarball.

Enjoy!