gSOAP for web services in C and C++

Mike Giles
Computing Laboratory

June 20, 2003

- web services
- gSOAP
- a simple example
- additional features

Web Services

Most web services operate in a client/server RPC (remote procedure call) manner:

- the server is running permanently (or at least is started first)
- the client makes a request to the server
- the server returns an answer

NOTE: unlike the Globus model, the server does not run under the client's userid, so the client does not have to have an account on the server machine.

Web Services

Within the basic client/server arrangement, there are a number of subtle distinctions.

A standard web server can offer services through CGI executables: it listens to port 80, and if a requests asks for a particular CGI to be executed to generate a response then it does it.

Alternatively, can have a standalone web service which listens to a particular port and deals with requests.

Web Services

What about handling multiple requests from different clients?

- could queue then up and process them one at a time
- could spin off a separate thread to deal with each one
- could spin off a separate process (through a *factory*?) to deal with each one

Web Services

What about handling multiple requests from the same client?

If the history of the interaction needs to be maintained (*persistence*), this can be done by opening a communication channel and maintaining it (*keepalive*) until the client closes it, or there’s a timeout.

(In this case, should use a separate thread or process for each client.)

Web Services

Standards are crucial for interoperability of web services.

SOAP (Simple Object Access Protocol) defines the RPC interaction:

- XML for the main content (request and response)
- optional MIME attachment (just like email)
- `http/https` to send the SOAP messages

There is no restriction on the choice of language for implementing the server or client application.

Web Services

Language-specific support for creating web services includes:

- Java: IBM Websphere, Sun ONE, Borland JBuilder, lots of others
- C#: Microsoft .NET
- Python: ZSI (Zolera Soap Infrastructure)
- C/C++: gSOAP

Web Services

There is also a standard (UDDI) for directories for

- publishing information about a service
- looking for services to carry out certain tasks

gSOAP does not address this aspect.

gSOAP

gSOAP is a package for generating web service servers and clients in C/C++ (and FORTRAN)

- a pre-processor generates additional C/C++ files given a header file specification of the RPC routines
- there are also some gSOAP files which contain the code to do all the conversion of data to/from XML
- the distribution includes 150 pages of documentation and lots of example applications

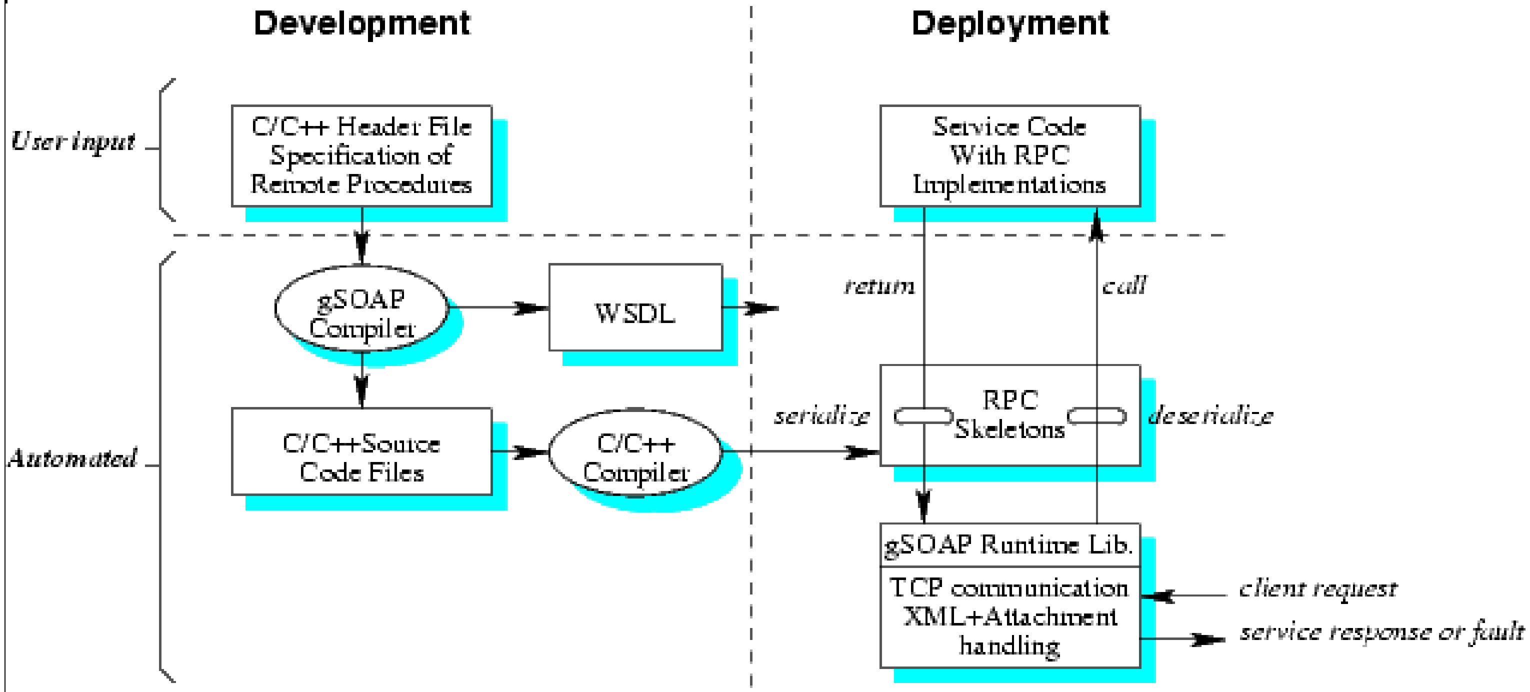
gSOAP

Evidence of the maturity of gSOAP includes:

- now developed through SourceForge after initial development at Florida State University
- IBM is using it as part of their Web Services Toolkit for Mobile Devices (maybe because of its low memory requirements?)
- reviews in Web Services magazines
- assessed as being very efficient by a rival project at Indiana University
- being used by GridLab, a major EU project

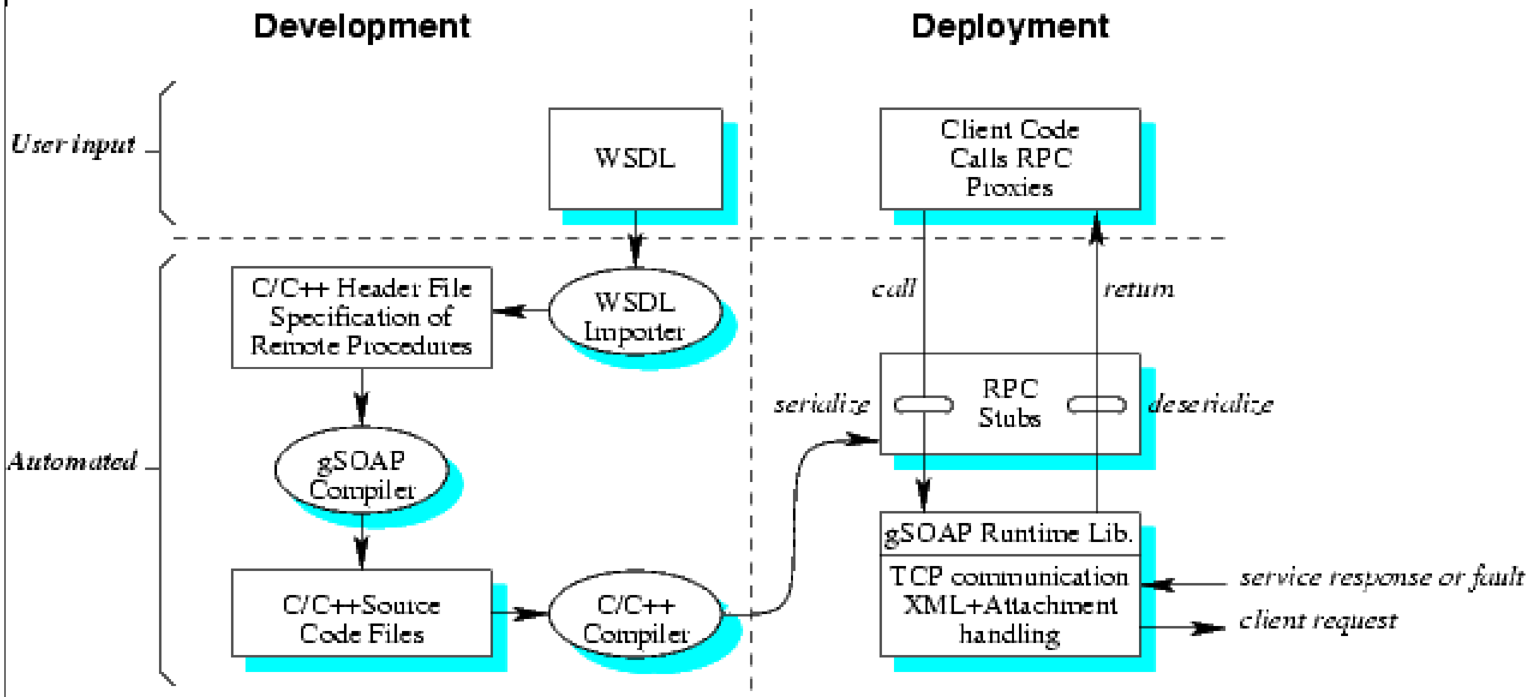
gSOAP

Server development and deployment



gSOAP

Client development and deployment



gSOAP

The example is a web service calculator which takes two numbers and adds or subtracts them.

For this application, the user writes 3 files:

- `calc.h`: a header file defining the RPC routines
- `calcserver.c`: the server code
- `calcclient.c`: the client code

calc.h

```
//gsoap ns service name:  calc
//gsoap ns schema namespace:  urn:calc

int ns__add(double a,double b,double *result);

int ns__sub(double a,double b,double *result);
```

The `ns__` prefix and the gSOAP declarations avoid ambiguities if an application needs to use two services with the same RPC names

calcserver.c

```
#include <math.h>
#include "soapH.h"
#include "calc.nsmap"

int main(int argc, char **argv)
{ int m, s; /* master and slave sockets */
struct soap soap;
soap_init(&soap);

m = soap_bind(&soap,NULL,atoi(argv[1]),100);

for ( ; ; )
{ s = soap_accept(&soap);
  soap_serve(&soap);
  soap_end(&soap);
}

return 0;
}
```

calcserver.c

```
int ns__add(struct soap *soap,
            double a, double b, double *result)
{ *result = a + b;
  return SOAP_OK;
}

int ns__sub(struct soap *soap,
            double a, double b, double *result)
{ *result = a - b;
  return SOAP_OK;
}
```

calcclient.c

```
#include "soapH.h"
#include "calc.nsmap"

const char server[] =
"http://bsaires.acm.caltech.edu:18083";

int main(int argc, char **argv)
{ struct soap soap;
  double a, b, result;

  soap_init(&soap);

  a = strtod(argv[2], NULL);
  b = strtod(argv[3], NULL);
```

calcclient.c

```
  switch (*argv[1])
  { case ’a’:
      soap_call_ns__add(&soap, server, "",
                       a, b, &result);
      break;
    case ’s’:
      soap_call_ns__sub(&soap, server, "",
                       a, b, &result);
      break;
  }

  if (soap.error)
    soap_print_fault(&soap, stderr);
  else
    printf("result = %g\n", result);
  return 0;
}
```

gSOAP

Additional features:

- multiple results handled by a result structure
- dynamic arrays handled by a structure with size and pointer
- *keepalive* for services needing persistence
- `https` and SSL for security
- Globus GSI *plugin* from GridLab group in Italy
- `zlib` and `gzip` compression
- MIME attachments

Conclusions

I think gSOAP may be very useful for eScience groups who primarily program in C/C++

- short learning curve
- easy to develop simple applications
- plenty of sophisticated features for more advanced use
- very efficient, with compression for reduced bandwidth
- `https`, SSL and GSI plugin for security
- main limitation is lack of any UDDI support