The developer guide to the scen_gen

Hongtao Ren, Marek Makowski, IIASA

Version 0.1, February 21, 2012

1 Scope of this document

This guide aims at supporting the developers in using scen_gen for the kernel testing, and in adapting the scen_gen to more specific needs. The document deals with the following issues:

- Functionality
- Architecture
- Structure and content of the part of the github repository containing the scen_gen
- Compilation of the scen_gen
- Installation of the scen_gen

2 Functionality of the scen_gen

The scen_gen is a simple c++ application developed for serving the following purposes:

- Illustrate how easy it is to consume the WSs (Web-services) from c++ applications
- Provide a template for other c++ applications consuming WSs for data management

Short characteristics of the scen_gen:

- gSoap it used for automatic generation of headers and DTOs (Data Transfer Objects)
- a WS is used for getting the SMS of the tiny test model (id == 71)
- the SMS is stored in the automatically generated DTOs
- the application just prints the DTO to the stdout

3 Overview the architecture

Figure 1 shows the WS-based interface between the Kernel and the scen_gen. The gSoap (http://gsoap2.sourceforge.net/) is used as the development toolkit. The gSoap toolkit takes the WSDLs and XML schemas, and map the XML schema types and the SOAP messaging protocols to easy-to-use and efficient C++ code (stubs, client proxy objects).

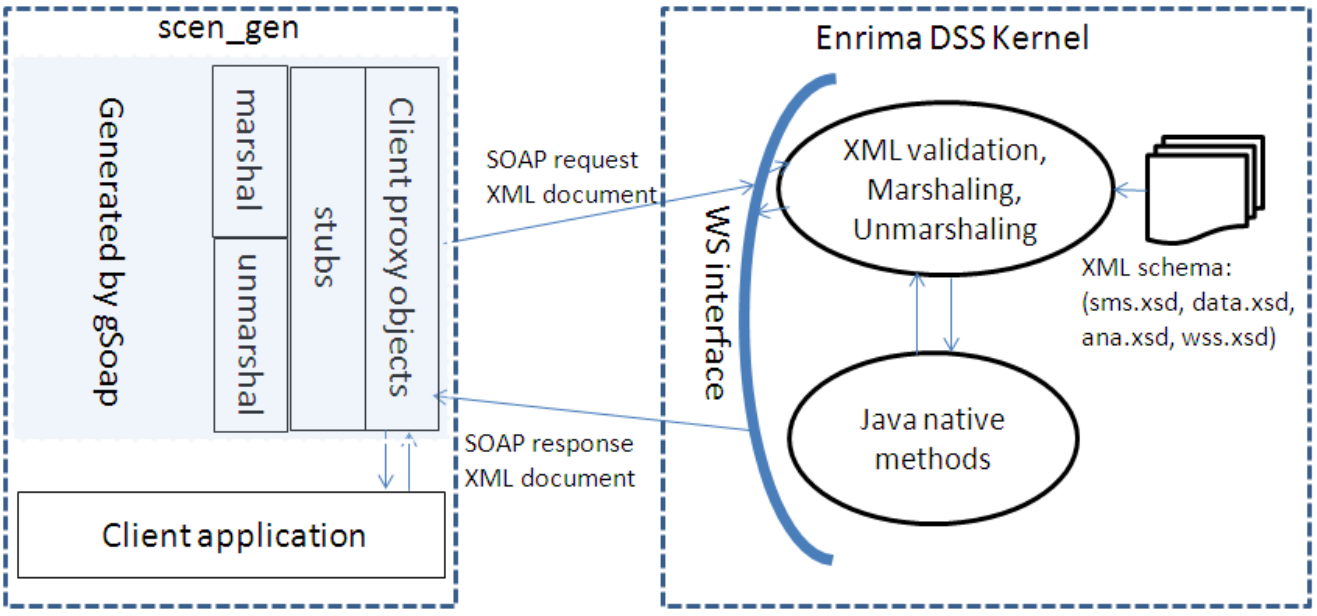

Figure 1 Kernel and scen_gen interface

The scen_gen contains c++ code generated by the gSoap and an illustrative small client application:

1) c++ code generated by gSoap: contains all stubs and proxy objects for consuming WSs
2) client application: provides business logic (get SMS, print SMS to the stdout, store entity values, etc.)

4 Files and directories

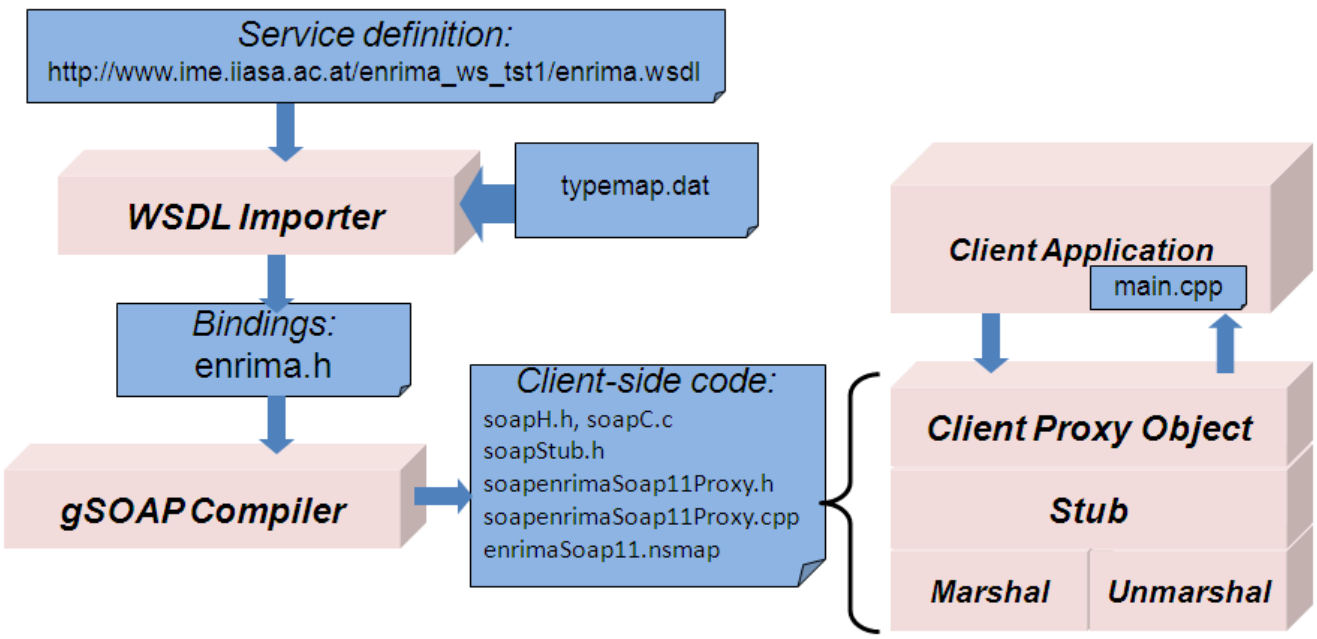

Figure 2 The structure of the files and applications

The structure of files and applications is illustrated in Figure 2; all the files but main.cpp are generated by gSoap toolkit automatically. The content of the files are as follows:

- stdsoap2.h, stdsoap2.cpp, typemap.dat: are generated by the gSoap.
- enrima.h, the intermediate header file specifies the bindings in an intuitive syntax. Primitive XML types such as built-in XSD types and XML schema simpleTypes are mapped to primitive C/C++ data types, and compound XML schema complex types are mapped to structs (for C) or classes (for C++). SOAP service operations are mapped to function-prototype declarations needed for implementation of the service proxies. The header is generated by the gSoap WSDL importer via the following command:
  $ wsdl2h -o enrima.h -t typemap.dat http://www.ime.iiasa.ac.at/enrima_ws_tst1/enrima.wsdl
- enrimaSoap11.nsmap, soapH.h, soapC.cpp, soapStub.h, soapenrimaSoap11Proxy.h, soapenrimaSoap11Proxy.cpp: all the stubs and client proxy objects generated by the gSoap compiler via the following command:
  $ soapcpp2 -Cx -i enrima.h -I $HOME/gsoap/share/gsoap/import/
- main.cpp
  A simple client application to illustrate how to invoke stubs and client proxy objects to get the SMS and print it to the stdout.

```
#include <iostream>
#include <fstream>
#include <vector>
#include "soapenrimaSoap11Proxy.h"
#include "enrimaSoap11.nsmap"
using namespace std;
enrimaSoap11Proxy enrimaSoap11Proxy(SOAP_XML_INDENT);
int getModel(int idModel, _ns1__getSMSResponse *ns1__getSMSResponse);
void printModel(ns2__modelSpec *model);
int main() {
        _ns1__getSMSResponse *response = new _ns1__getSMSResponse;
        int returnCode = getModel(71, response);
        if (returnCode == SOAP_OK)
        {
                printModel(response->modelSpec);
        }
}

int getModel(int idModel, _ns1__getSMSResponse *ns1__getSMSResponse) {
        _ns1__getSMSRequest *req = new _ns1__getSMSRequest;
        req->idModelSpec = idModel;
        if (enrimaSoap11Proxy.getSMS(req, ns1__getSMSResponse) == SOAP_OK)
        {
                return SOAP_OK;
        } else {
                enrimaSoap11Proxy.soap_stream_fault(cerr);
                return enrimaSoap11Proxy.error;
        }
}
void printModel(ns2__modelSpec *model) {
        cout << "model id: " << *model->id << " name: " << model->status <<
endl;
        for (unsigned short i = 0; i < model->setSpec.size(); i++) {
                ns2__setSpec *setSpec = model->setSpec.at(i);
                cout << "setSpec id: " << *setSpec->id << ", label:" <<
*setSpec->name<< endl;
        }
        for (unsigned short i = 0; i < model->entitySpec.size(); i++) {
                ns2__entitySpec *entitySpec = model->entitySpec.at(i);
                cout << "entitySpec id: " << *entitySpec->id << ", label:"<<
*entitySpec->name << endl;
        }
}
```

Figure 3 code of the main.cpp

5 Compilation of the scen_gen

5.1 Environment

- C++ compiler
- Git (optional)

5.2 Download the Kernel repository

$ git clone git://github.com/enrima-dev4/kernel.git

If the git is not available, then one can interactively zip the Kernel repository posted at https://github.com/enrima-dev4/kernel, and download the zipped repository.

5.3 Download and Install gSoap toolkit

- download gsoap from http://sourceforge.net/projects/gsoap2/files/gSOAP/, please select gsoap_2.8.3.zip or above
- unzip and install gsoap (assume you unzip it to $HOME/gsoap-2.8)
- install gSoap
  $ cd $HOME/gsoap-2.8
  $ ./configure —prefix=$HOME/gsoap && make && make install

5.4 Generates intermediate headers, stubs client proxy objects

- create project scen_gen:
  $mkdir scen_gen
  $cp $HOME/gsoap-2.8 /gsoap/{typemap.dat,stdsoap2.{h,cpp}} scen_gen
- generate enrima.h
  $HOME/bin/wsdl2h -o enrima.h -t typemap.dat
  http://www.ime.iiasa.ac.at/enrima_ws_tst1/enrima.wsdl
- generate DTOs
  $HOME/bin/soapcpp2 -Cx -i enrima.h -I $HOME/gsoap/share/gsoap/import/

5.5 Compile and run

- Copy the main.cpp file the project folder
- linux: $g++ -o enrima *.cpp or unix: $g++ -o enrima *.cpp -lsocket -lnsl
- ./enrima