

SOAP Based Distributed Simulation Environment for System-on-Chip (SoC) Design

Samy MEFTALI Anouar DZIRI Luc CHAREST Philippe MARQUET

Jean-Luc DEKEYSER

Laboratoire d'Informatique Fondamentale de Lille

Université des Sciences et Technologies de Lille

France

Abstract

In this paper we present a distributed simulation environment for System-on-Chip (SoC) design. Our approach enables automatic generation of, geographically distributed, SystemC simulation models for IP-based SoC design and eases communication between heterogeneous parts of the simulation. The SystemC simulation model follows a client/server architecture, where each client/server should contain some IPs composing the system. The number of distributed SystemC simulators used in the simulation platform is not a priori limited a priori. The communication between them is made using Simple Object Access Protocol (SOAP) through communication standards as XML and HTML. The feasibility of our method is shown by an example composed by four SystemC modules, simulated on three hosts.

1 Introduction

The growing complexity of SoCs, which integrate software parts as well as specific hardware parts (IPs, Intellectual Property), makes their validation and verification a highly time consuming process. Generally, the design process needs a seamless collaboration between different design tools and teams that are often specialized and geographically distributed. In order to validate the overall system, distributed simulation environment becomes a key issue to allow significant reuse of CPU resources and human competences. Thus, a user can simulate his complete system on geographically distant hosts. Moreover, complex systems-on-chip require more and more IP components coming from third part vendors. However, before purchasing IPs, designer needs to evaluate the performance of the entire system (functionality, speed, area, power consumption, etc.). Indeed, the distributed simulation allows IP vendor to expose their IPs and hide in the same time behavior details.

Generally, such simulation environment is based on a client-server architecture. This later has been employed by many Internet-based applications. However, there is still a lack for tools allowing distributed simulation of SoC. Moreover, existing approach uses middleware (such as .Net, CORBA, RMI, Socket, etc.) for the communication between IPs on client and server parts. Unfortunately, communication approach based on these middleware stills suffering from many lacks, mainly compatibility and security (firewalls and proxy servers will usually block this kind of traffic). Moreover, the cost of middlewares (such as CORBA, .Net) is still expensive and implementing solutions around them presents a tedious human work.

The main goal of our current works is to enable automatic generation of SystemC [?] wrappers and to ease communication between different IP components involved in the simulation process by using Simple Object Access Protocol (SOAP). This communication will be only based on HTML and XML standards.

The remaining of this paper is structured as follows. The next Section presents some necessary basics to understand our methodology. In Section 3 we give a review of state of the art on distributed simulation, and our contribution. The different steps of our methodology are presented with details in Section 4. In Section 5, a case study illustrating the use of our approach is presented. Section 6 summarizes and concludes the paper.

2 Basics

We define in this section the main concepts used in our simulation framework: SOAP and distributed simulation.

2.1 Distributed Simulation in SoC Design

We mean by a distributed simulation an executable model of a given system on two (or more) concurrent simulators running on distant hosts. Thus, each module or IP (processor, memory, etc.) of the SoC may be executed on a given host as shown in figure ???. For some commercial or licenses constraints, some IPs must be executed on a given host and the designer has all the freedom for choosing the hosts (from the available hosts) on which the remaining IPs will be executed.

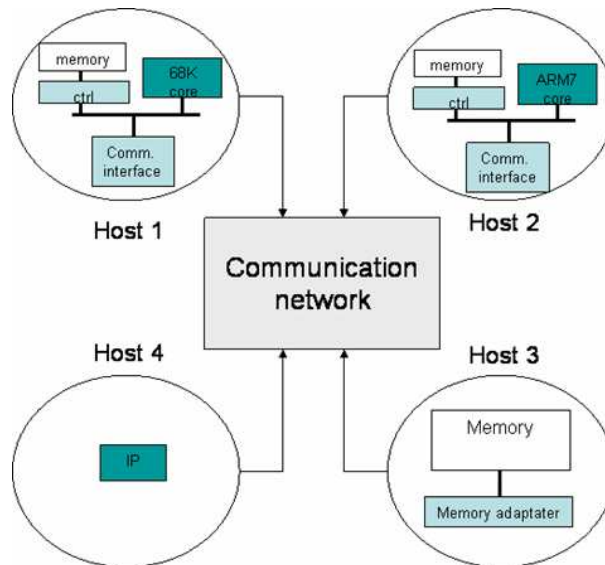


Figure 1: A SoC distributed simulation on distant hosts

2.2 SOAP

The task of creating and deploying web services is really not all that difficult, nor is it all that different than what developers currently do in more traditional web applications. Most programmers don't need to know the exact details of encodings and envelopes; instead, they'll simply use a SOAP toolkit such as those described here.

Today's client-server based applications communicate using Remote Procedure Calls (RPC) between objects like DCOM, RMI and CORBA. The tendency on all platforms is to automate more and more of the minute details and tedious work in handling communication and creating web services. However, RPC represents a compatibility and security problem; firewalls and proxy servers will normally block traffic between client and server parts. A better way to communicate

between applications is over HTML, because HTML is supported by any platform. SOAP, stands for Simple Object Access Protocol, was created to accomplish this. It also avoids to programmers to have any knowledge about details of encodings and envelopes.

SOAP (figure ??) [?] is an XML based protocol that provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. It is a platform independent lightweight protocol for exchanging information in a decentralized, distributed environment using HTML.

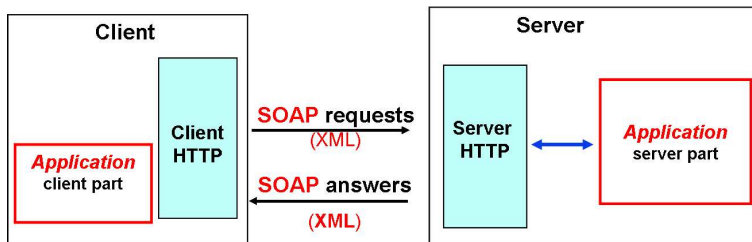


Figure 2: SOAP for distributed applications

3 Related Work

We find in the literature several tools allowing the geographically distributed applications. The communication between distributed parts is based on different concepts called middleware. We can identify two kind of communication levels: a low level communication (using Sockets), and a high level one that is used to enable communication between distributed objects (using middleware such as CORBA [?], .Net, RMI, and SOAP).

MCI [?, ?] is a multilanguage co-simulation tool that uses sockets to enable communication between different modules of a SoC. These modules can be described in VHDL and in C language, the SoC can then be simulated on simulators residing in geographically distant hosts, using a co-simulation backplane that is implemented onto UNIX Sockets. MCI allows the use of more programming languages such as SDL and MATLAB.

Synthesisia [?] is a HW/SW co-simulation tool allowing simulating together 1–the hardware part of a SoC on a VHDL simulator and 2–the software part as compiled C or ADA programs. It uses Sockets to enables communication between different tasks executed on many hosts.

CoWare proposes a simulation environment [?] permitting the distributed simulation of C/C++ and VHDL code. The communication between modules is realized using Sockets.

COSSAP [?] allows simulating simultaneously and concurrently modules described in C and assembling code with VHDL modules. It uses UNIX (IPC/Sockets) for the communication between modules.

&Sim is a flexible simulation environment, where two or many simulators may run concurrently and communicate via a network using CORBA bus (ORB) [?]. Different tools, such as Statemate and Simulink, are supported by this co-simulation environment.

[?] describes an automatic distributed co-simulation framework based on sockets communication. This environment allows easily multilanguage simulations, but unfortunately it is strongly related to operating systems used on the different simulation hosts.

eSYS.net [?] is a simulation environment for SW/HW design, based on .Net for the communication. IPs composing the system are described using C# language, but this tool do not supports distributed simulation models.

Some other tools exist and follow a Web Service architecture (communication between client and server is done using SOAP) such as in [?, ?, ?]. However, this type of architecture is still

used by the software community and it does not yet allow the construction of applications for hardware/software co-simulation.

3.1 Contribution

Our main contribution is the use of web services for communications in distributed SoC simulation. The originality of our work is the genericity in terms of languages (we use in this paper SystemC as an example, but the methodology still working using any other language), but also the encapsulation of communications inside standards (HTML, XML).

4 Distributed Simulation Model Generation

4.1 Distributing SystemC

The standard SystemC [?] does not allow at all distributed simulations on a network. In fact, it has no standard dedicated API for communications over a network. Thus, in order to make it possible, we defined a new concept called distribution wrappers. It is presented with details in the remaining of this section.

4.2 SOAP Based SoC Simulation

The main idea of this work is to permit to designers to use IPs available on remote hosts of different providers. So we must be able to simulate a system designed using distributed IPs. We could use for designing a system, some IPs distributed on several providers. Thus, our solution is able to simulate a system using several SystemC simulators communicating by exchanging SOAP messages HTML and HTTP protocol for instance.

The principle illustrated in figure ?? consists in integrating in our system distant IPs. The use of a web registry service (like UDDI which will be used as a directory where providers can register and search for IP services) can be used by clients application to dynamically discover different services provided by a given distant IP (end point URL). Communication between different IPs is made by term of services. Indeed, services are described by using WSDL (Web Service Description Language). Thus the tool is able to locate a distant IP having just its provided services.

This mechanism is completely transparent for the designer. Thus to design a SoC, if we want to use distant IPs, it will be enough to use some specific mechanisms described in the next paragraphs.

4.3 Distribution wrappers

A distributed wrapper encapsulates each client or server involving in the simulation. It has two objectives, and thus it is composed by two parts. It has a process synchronizing the simulation (simulators), and a SOAP interface transforming inter-modules C++ (SystemC) data types into XML (SOAP) messages using available generic toolkits.

4.4 SOAP interface (API)

These web services development toolkit offers an XML to C/C++ language binding to ease the development of SOAP/XML web services in C and C/C++. Most toolkits for C++ Web services adopt a SOAP-based view and offer APIs that require the use of class libraries for SOAP-specific data structures. This often forces a user to adapt the application logic to these

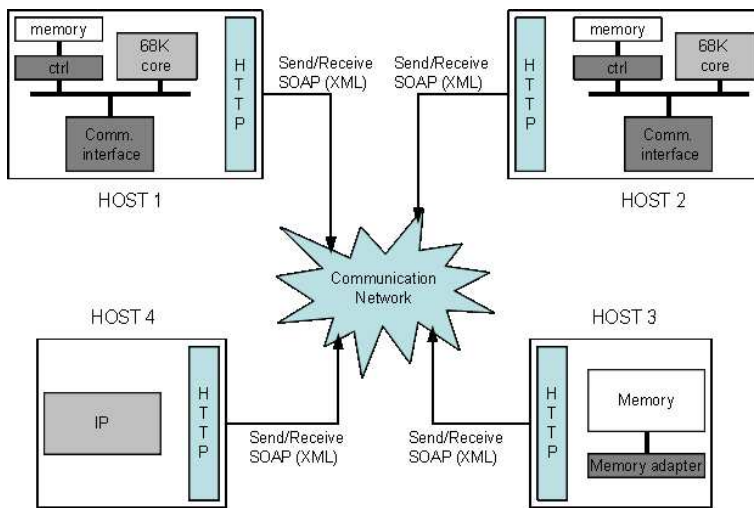


Figure 3: SOAP based SoC simulation

libraries. In contrast, generic toolkits as gSOAP provides a transparent SOAP API through the use of proven compiler technologies. These technologies leverage strong typing to map XML schemas to C/C++ definitions. Strong typing provides a greater assurance on content validation of both WSDL schemas and SOAP/XML messages. The gSOAP is an example of compilers able to generate efficient XML serializers for native and user-defined C and C++ data types. As a result, SOAP/XML interoperability is achieved with a simple API relieving the user from the burden of WSDL and SOAP details, thus enabling him or her to concentrate on the application-essential logic.

4.5 Synchronization Process

Our distributed wrappers can encapsulate several IPs at the same time. Indeed, there is only one envelope per distant simulator. The other characteristic of these wrappers is that they always reference a method in a SOAP interface. There are two types of distributed wrappers. A wrapper called `server_if` for the server side and an other one called `client_if` for the client side. The two types are closely similar, they encapsulate IPs in the same way. In fact, Both have references to the SOAP interface generated by the compiler. We specified previously that the client synchronizes and "arbiters" the simulation. Indeed, it is the client who carries out the calls of distant methods. When the client wrapper is updated by the SystemC simulator of the same client, the wrapper executes an algorithm. In our solution we focus for the moment on systems having a clock, each top of the clock may be an event. Moreover, we use clock signal to synchronize the simulators. Here is the algorithm executed by the client's wrapper after each update. If an event occurs (a clock signal (for example)):

1. call for each input port of the client wrapper the method `setValue()` generated by the soap toolkit compiler.
2. run the simulation on the server side for one period equal to those of the clock by invoking the method `doSim()`
3. get the values output port of the client wrapper with the method `getValue()` generated by the soap toolkit compiler.

The server's wrapper is updated by the calls of distant methods carried out on its SOAP interface. Thus, when calling the method `setValue()` of the SOAP API, the server's wrapper

updates the value of the port passed, to the method, as argument. In the same way, when calling the method `getValue()`, the server's wrapper transmits the value of the port passed as parameter. In our methodology we can qualify the client simulator as a main (master) simulator and qualify the simulators of the servers as slave ones. This qualification is due to the fact that the slave simulator executes all the synchronization operations.

4.6 Summary

Simulating a given SoC specification using our methodology involves the following steps:

- decide for each IP on which host it will be simulated
- group the IPs in a set of servers and clients
- generate SOAP interfaces for each client and each server
- generate a distributed wrapper for each host
- run all servers
- each server registers its services in the services register (UDDI), then waits for a client request
- run the client (simulation master)
- the client asks the UDDI for all services he needs for the simulation
- the UDDI furnish to the client the localization (address) of a server corresponding to each one of services he need
- the client starts the simulation (`doSim()`), and synchronizes all servers.

5 Application

As an application example, we use a system composed by three IPs (figure ??). The first one produces data (`sc_product`), the second calculates the square of the produced data (`sc_square`), and the last IP stores the square (`sc_consum`).

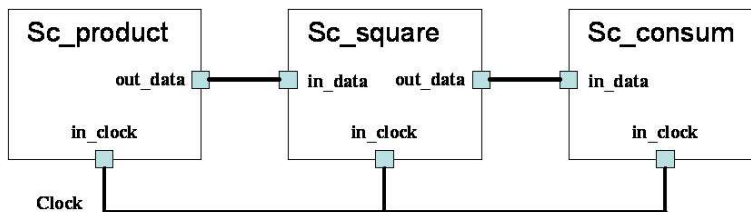


Figure 4: Application example

The repartition of theses IPs on the three available hosts for the simulation is:

- `sc_product`: Client
- `sc_square`: Server A
- `sc_consum`: Server B

This system (figure ??) is synchronized by a global clock with a period of 10 nanoseconds. Thus all the IPs are connected to this clock. We choose this particular application case to avoid the problems related to difference in terms of simulation cycles of remote simulators on the simulation hosts.

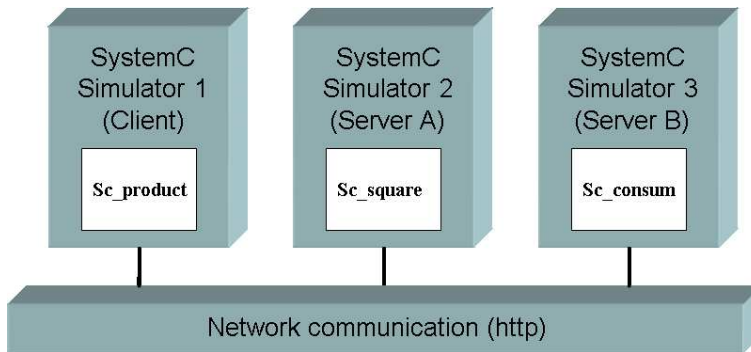


Figure 5: Simulation of the example on a platform composed by three SystemC simulators

5.1 Services Registration

The two servers register their services (methods) on the UDDI. Figure ?? shows for instance the registration file of the square service (server A). All information allowing clients to use this service are given in this file. In fact, service name, access port, service style and the service namespace are given.

```
// Contents of file « Square.h":
//gsoap ns service name: Square
//gsoap ns service style: rpc
//gsoap ns service encoding: encoded
//gsoap ns service port: http://mydomain/path/square:80
//gsoap ns service namespace: urn:Square
double ns__Square(double a, double b);
```

Figure 6: Services description of host 2

5.2 Results and Analyses

We easily see in this application example that the encapsulation of the three IPs composing the system, and the transformation of their interfaces code is very simple. These transformations are entirely automatic. This application is described at a Transactional Level (TL), thus the simulation time on the distributed platform is nearly equal to simulation time on a single simulator. But, in the case of an RTL model, the distributed simulation can be very time consuming because of the transactions over the network. In the case of RTL simulation, our methodology is still useful if some IPs must be simulated on the provider's host without downloading them. The lack that we can note is the important amount of the added code to the initial specification. This additional code is mainly due to the encapsulation of the exchanged data in XML/html code. But this new code is generated automatically and is never directly handled by the designer.

6 Conclusion

In this paper we presented a novel methodology allowing the SystemC simulation on distributed hosts. In our approach the different simulators communicates using the SOAP communication protocol. This communication mean make the simulation environment very easy and flexible and free designer from a tedious manual work. The fact that SOAP supports the heterogeneity (in terms of languages and platforms) will permits to integrate easily some other description languages, to our simulation platform in the future.

References

- [AA04] El Mostapha Aboulhamid and Al. eSYS.net. <http://www.esys-net.org/home.html>, 2004.
- [Alt95] M. Altmae. Hardware/Software Coverification. In *EDA Traff*, March 1995.
- [CoW] CoWare inc. <http://www.coware.com/cowareN2C.html>.
- [GLA02] T. Grotker, S. Liao, and Al. *System Design with SystemC*. Kluwer Publishers, 2002.
- [Goo] Google. Google Web APIs. <http://www.google.com/apis/>.
- [HCNJ00] F. Hessel, P. Coste, G. Nicolescu, and A. Jerraya. Multi level communication synthesis of heterogeneous multilanguage specifications. In *ICCD 2000*, Texas, USA, September 2000.
- [HLV⁺99] F. Hessel, Ph. LeMarec, C. Valderama, M. Romdhani, and A. Jerraya. *MCI - Multilanguage Distributed Co-Simulation Tool*. Kluwer, 1999.
- [MVD03] Samy Meftali, Joel Vennin, and Jean-Luc Dekeyser. Automatic generation of geographically distributed system simulation models for IP/SoC design. In *The 46th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS 03)*, Cairo, Egypt, December 2003.
- [Obj01] Object Management Group, Inc., editor. *Common Object Request Broker Architecture (CORBA), Version 2.6*, chapter 24, Real-Time CORBA. <http://cgi.omg.org/cgi-bin/doc?formal/01-12-62.pdf>, December 2001.
- [OMG05] OMG. W. S. M. http://www.omg.org/06-4_Final_Prism_Web_Services_Messaging.pdf, 2005.
- [Ope02] Open SystemC Initiative. SystemC. <http://www.systemc.org/>, 2002.
- [RC95] P. Runstadler and R. Crevier. Virtual prototyping for system design and verification. Technical report, Synopsys Inc., 1995.
- [YMYG04] Dean Yao, Tisson Mathew, Mazin Yousif, and Sharad Garg. A Framework for Platform-Based Dynamic Resource Provisioning. Technical report, Research and Development at Intel, 2004.
- [ZZ04] Gongxuan Zhang and Guowei Zuo. A SOAP-Based Electronic Commerce Security Mechanism. *Parallel and Distributed Processing Techniques and Applications*, 2004.