CORBA Communication Backplane for Design and Verification

Pascal Giard, Jean-François Boland, Jean Belzile
Laboratoire de Communication et d'Intégration de la MicroÉlectronique (LACIME),
Department of Electrical Engineering, École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, CANADA H3C 1K3
Tel.: 514-396-8800 ext. 8426, Fax: 514-396-8684
Emails: pascal.giard@lacime.etsmtl.ca, {jean-francois.boland,jean.belzile}@etsmtl.ca

***Abstract*—This paper presents a novel object based communication strategy to interconnect design tools and system components operating at various abstraction levels to produce a consistent and coherent system. Interconnects are generated from interface models. This platform can be used to prototype, validate, simulate and/or test the complex systems interactions before all the components are built and integrated. Results show a 20% increase in communication overhead compared to MathWorks Link for ModelSim while providing a generic solution over the tightly coupled offering from MathWorks.**

***Index Terms*—CORBA, backplane, heterogeneous, co-simulation, verification.**

I. Introduction

Integrated circuit design complexity increases at an impressive rate forcing designers to introduce heterogeneousness in the design flow as an attempt to tackle the productivity gap. *System-on-Chip (SoC)* is a good example of design complexity where heterogeneous *intellectual property (IP)* cores are integrated together. IP cores are designed using different modeling paradigms according to intrinsic characteristics. As a result, specific tools and languages are used at different stages in the design flow. Moreover, an IP core may be described at different abstraction levels during the design process where each abstraction level often has its own modeling paradigm.

Verification of such heterogeneous systems implies running multiparadigm models together. This heterogeneity makes design verification an expensive task. Experts agree that functional verification consumes anywhere between 50% and 75% of the design resources (time and effort)[1].

A good vehicle to implement heterogeneous systems is a virtual functional prototype, which is an executable specification of the system that can be used to master the algorithms with a variable degree of architectural constraints. When it comes to verification, heterogeneous systems require complex mechanisms to achieve intercomponent communication. Traditional solutions provided by *commercial off-the-shelf (COTS)* tools are twofold. The first one is to internally support a limited subset of modeling languages representing multiple levels of abstraction. This method lacks flexibility and relies on the company willingness to provide support for given modeling languages. The second one, often called "ad-hoc coupling" provides a specific - often closed - way to interface to a simulator.

In this paper, we propose an architecture where the actors involved communicate within a *Common Object Request Broker Architecture (CORBA) i.e.* a common communication backplane as proposed in [2]. Section 2 discussses related work, section 3 presents the general architecture, section 4 shows our implementation, section 5 presents the results and section 6 gives our conclusions.

II. Related Work

Much research work have been done to try to overcome the challenges of simulation-based verification of heterogeneous systems. Unfortunately, in most cases the problem has not been entirely solved.

The authors in [3] present the necessity of multiparadigm modeling in embedded systems design. Embedded systems, like *electronic control unit (ECU)*, are also comprised of heterogeneous components. According to [3], the description of the different aspects and views of the whole system and subsystem requires corresponding modeling paradigms. While the designs intent is different, the necessity of multiparadigms modeling results in a similar problem: the integration of several languages and tools. As in [3], currently available tools used in complex *integrated circuit (IC)* design flow provide insufficient support and strategies for multiparadigm modeling.

H.D. Patel et al. proposed the CARH architecture[4] which uses an *Object Request Broker (ORB)* to create a validating environment for generated SystemC models. CARH integrates CORBA and uses an ORB as a generic communication medium allowing great flexibility and expandability. However, this architecture is aimed at a different scope *i.e.* it uses CORBA to observe components' behavior. Moreover, it is restricted to the SystemC language and requires modifications to the *Open SystemC Initiative (OSCI) Simulator*.

The novel distributed object communication architecture presented in this paper uses a tool integration mechanism based on process flows using ORBs as described in [5]. The following section presents the proposed architecture.

III. Architecture

This section describes key factors of our verification methodology namely the architecture model, the communication framework and tools involved in our heterogeneous design flow. Tools are presented in three subsections: tool adaptors, component wrappers and client/server wrappers.

A. Architecture Model

CORBA is a distributed object architecture standard by the Object Management Group (OMG) widely used in a large spectrum of applications ranging from low level military communication devices to high level free software computer applications. Object's interfaces are expressed using the *Interface Definition Language (IDL)*.

Objects can be implemented in various languages and on various platforms. Actually, CORBA is platform and language agnostic. Client and server wrappers are automatically generated from IDL files. Current language mappers can generate CORBA objects in many languages such as Ada, C, C++, Java, Eiffel, Python, PL/1 and VHDL.

Moreover, CORBA has an extensible transport framework. Thus, CORBA is not limited to one transport protocol *i.e.* pluggable transport implementations can be added.

B. Communication Framework

Communication is one of the biggest challenges when verifying heterogeneous designs. The rationale behind the usage of a distributed object communication architecture as a backplane is multifold. Verification with components implemented at various levels of abstraction often implies that multiple languages and simulators have to interact together. Early hardware verification with both development boards and hardware prototypes is an asset. Data analysis that does not require any modification to the design or any other time consuming task is nice to have. CORBA allows us to achieve all this at the cost of interfacing the tools involved with the architecture. A simplified overview of the architecture is presented in figure 1.

Each simulator or tool requires an ORB which handles intercommunication. The simulator or tool acting as the *Master* has one to many CORBA clients. The other simulators or tools act as servers exposing their respective components via the CORBA *Naming Service*.

Components included in the *Master* but not simulated in the latter are proxy components. In fact, proxy components are CORBA clients sending requests to CORBA servers which in turn, talk to the real components. The *Master* believes those proxy components are native components. Proxy components are one form of tool adaptors.

C. Tool Adaptors

Almost each and every single simulator offers an external interface one can hook up to. The tool adaptor uses that external interface to make the link between CORBA objects and simulated components. Its jobs may include starting, pausing, resuming and stopping simulation as well as providing means for the component wrappers to initialize a component and most importantly put or get signals.

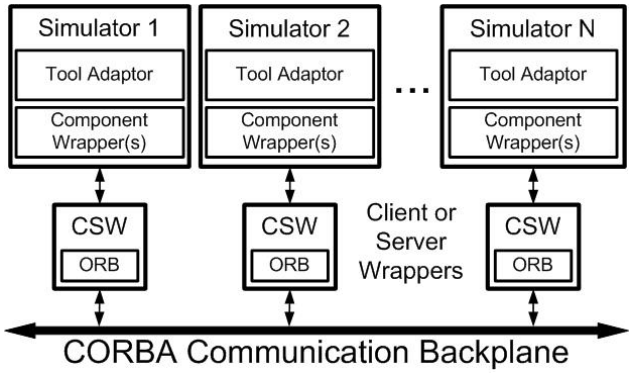

Fig. 1. CORBA Communication Backplane.

In some cases the tool adaptor is only a container or a shared library in which the component wrapper is implemented *e.g.* ModelSim Foreign Language Interface (FLI) or GHDL VHDL Programmable Interface (VHPI) or even Simulink S-Functions.

D. Component Wrappers

Going from an application domain to another sometimes requires functional or data adaptation. In those cases, a component wrapper acts as a transactor. Standing between the native component and the CORBA client, the component wrapper exists for this sole purpose. A Simulink block, a SystemC module or a VHDL design are examples of components.

In case where no functional or data adaptation is required, the component wrapper simply passes on signals from the component to the client or server and vice versa. Before reaching the client or server, signals go through both the tool adaptor and the client or server wrapper.

E. Client or Server Wrappers

Taking care of initializing, running and destroying the CORBA objects, the client and server wrappers also pass on messages received from ORBs to component wrappers and vice versa.

Moreover, the current client and server wrappers implementation permits users to configure CORBA endpoints at runtime.

The electronic design field contains a plethora of modeling languages and tools each of which excels in its respective application domain. This design environment encourages designers to use the right tools for the right tasks. As communication within the design environment is done with ORBs, once a tool is interfaced with an ORB, it can be integrated to the architecture.

Thus, one could use SystemC models running in the OSCI Simulator for system design, VHDL running in *Mentor Graphics ModelSim* for RTL and *MathWorks MATLAB and*

Simulink for data generation and result validation against a golden model.

The next section proposes implementations for this use case. Current implementation is multiplatform. MATLAB/Simulink runs under Microsoft Windows XP while the OSCI Simulator, Mentor Graphics ModelSim and the CORBA Naming Service run under GNU/Linux.

IV. Implementation

As mentioned in the previous section, the proof of concept implementation reproduce a tradional verification flow as shown in figure 2. Hence, each actor needs an ORB. We use a free software implementation: *The ACE ORB (TAO)*. Components interfaces are expressed with the IDL to be mapped to a programming language. Mapping tools such as *tao-idl* autogenerate interconnects: CORBA objects *a.k.a.* ORBs.

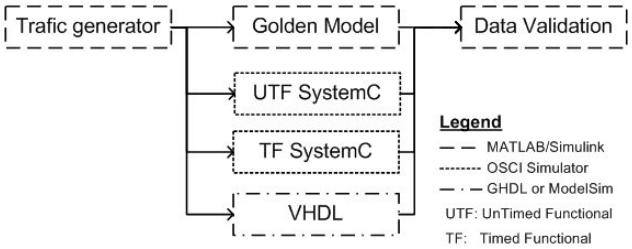

Fig. 2. Proof of concept verification environment.

ORBs either acting as a server or a client require a wrapper, component wrappers and ultimately tool adaptors. Most client and server wrappers and component wrapper functions being trivial to implement, in the following subsections, emphasis is put on tool adaptor implementations.

A. MathWorks MATLAB/Simulink

MATLAB/Simulink is an algorithmic level modeling language and numerical computing environment used to simulate dynamic systems. It here acts as the *Master* therefore it only has clients: support for simulation flow control is not required. The tool adaptor is implemented using level 2 MEX-files: S-Functions. S-Functions provide a mechanism for extending Simulink via callback methods.

Simulink is data driven and a method, `mdlOutputs` , is called every time new data comes in. That is where signals are passed on to the client wrapper and, if required, data and functional adaptation takes place. Proxy component outputs are updated before leaving the `mdlOutputs` method.

See code listing 1 for an example where simple data adaptation is carried out using macros.

```
#define fix2float(v) (((real_T)(v))/(1024*1024))
#define float2fix(v) ((int32_T)(v*1024))

static void mdlOutputs(SimStruct *S, int_T tid)
{
  // Retrieve C++ object from the pointers vector
  InputRealPtrsType u = ssGetInputPortRealSignalPtrs(S,0);
  real_T *y = ssGetOutputPortRealSignal(S, 0);
  CommLink *c = (CommLink *) ssGetPWork(S)[0];
  if(c->orb != 0) {
    try {
      // Send data to FirRtl server
      CORBA::Long output = 0;

      bool result = c->orb->update( float2fix(*u[0]), // dataIn
                                    *u[1],            // setup/run
                                    *u[2],            // number of taps
                                    output );         // dataOut
      // Update S-Function output
      y[0] = fix2float(output);
    }
    catch(const CORBA::Exception& ex) {
      // Handle communication error.
    }
  } else {
    // Handle initialization error.
  }
}
```

Listing 1. Extract from MATLAB/Simulink component wrapper and tool adaptor.

B. Open SystemC Initiative Simulator

The OSCI Simulator supports C++ natively and offers mechanisms to control the simulation flow. Thus, it is straightforward to merge the component wrapper and tool adaptor.

Simulation flow is controlled using the `sc_start` method. Signals can be accessed directly. In case where data require adaptation - *e.g.* conversion from double to fixed-point arithmetic - it is carried out using standard C++ functions.

C. Mentor Graphics ModelSim

ModelSim is a hardware simulator and debugger supporting Verilog, SystemVerilog, SystemC and VHDL. It has a VHDL *Foreign Language Interface (FLI)* which allows getting and setting values in VHDL objects. To a certain extent it also provides control over simulation. C/C++ code using FLI are compiled and used as shared libraries.

At elaboration time, drivers are created for each signals pushed to a design and a VHDL process is created using `mti_CreateProcess` . That process is first executed at time 0 allowing design initialization including the first call to `mti_ScheduleWakeup` . Following executions are triggered at times specified by previous calls to `mti_ScheduleWakeup` .

Since the FLI interface does not allow full simulation flow control, interaction with CORBA objects is done using POSIX mechanisms. The CORBA object and server wrapper runs in a different process. POSIX semaphores and shared memory are used for synchronization and data exchange.

D. GHDL

GHDL is a complete free software simulator for VHDL using GCC. Unlike the OSCI Simulator, it is impossible to control GHDL's simulation flow. Therefore, the POSIX semaphores and shared memory mechanisms are used to integrate the CORBA backplane to GHDL via the VHPI interface.

Despite its incompleteness, GHDL's VHPI implementation allows interaction with designs via callbacks alike

ModelSim's FLI. A function is registered against the `cbReadOnlySync[h]` callback. That function is then called each time a signal gets updated.

The first time the `cbReadOnlySync[h]` callback is triggered, a function which initializes the design is called. Successive calls are used to get or put values as desired. Signals are read using `vpi_get_str` and driven using `vpi_put_value` .

Although clients and servers are autogenerated, communications between ORBs are of blocking type. When a server receives a message from a client, it reads the message, updates the component's input signals, simulates the required number of clock cycles, sends back output signals and waits for another message from a client.

Similarly, a client sends a message to a server, waits for an answer, updates its output when the message is received and continues its operations. Messages sent by clients include input signals for design components. Messages received by clients include output signals from design components.

V. Results

This section presents a subset of the results that we have obtained with our architecture. Results were obtained using two computers connected to a dedicated 1000MBit TCP/IP network. An Intel Xeon E7525 3.6GHz with 3GB of RAM was running MathWorks MATLAB/Simulink 7.4.0 on Microsoft Windows XP SP2 and a dual Intel Quad Core Xeon E5405 2.0GHz with 16GB of RAM was running the Naming Service along with all simulation slaves on Ubuntu GNU/Linux 8.04.

As table I shows, a performance comparison between our CORBA Communication Backplane and MathWorks Link for ModelSim product reveals a difference that does not exceed 20% when simulating a 40 taps FIR filter.

TABLE I
Performance comparison between MathWorks Link for ModelSim (L4M) and CORBA Communication Backplane (CCB).

Simulation Time(s)	Real Time (s)		Difference
	L4M	*CCB*	
2,5	12,125	13,582	12,02%
5	23,493	26,865	14,35%
7,5	34,677	40,203	15,94%
10	45,995	54,761	19,06%
15	68,362	80,884	18,32%
25	113,285	135,188	19,33%
40	181,276	216,591	19,48%
55	248,892	296,968	19,32%
70	316,247	377,368	19,33%
85	384,831	459,675	19,45%
100	451,637	538,745	19,29%

Although this communication overhead is significant, it may become negligible where communication between design entities is minimal and design simulation requires heavy processing. Moreover, our generic design and verification architecture enables a broader tool base to work together resulting in seamless integration of different modeling languages and abstraction levels.

With its components wrappers that can act as transactors, this verification environment gives the designer the ability to do targeted design component refinements. He can express a design component in another formalism or move it on a different networked computer and still be able to verify the design as a whole as if design components were all expressed at the same level of abstraction and running in the same simulator. Besides, the designer can point component proxies to other component implementations by changing a string in the Simulink block properties, thus enabling quick architecture exploration.

VI. Conclusion

This paper's main contribution is a novel and generic design and verification architecture based on a distributed object communication architecture. Our work differs from [4] as we use CORBA not only to observe but to interact with them. Additionally, integration with the OSCI Simulator does not require any modification to the upstream code base.

Our design and verification architecture enables cosimulation of components expressed at different abstraction levels and in different modeling languages, running in different simulators or platforms, along with flexible tools integration as shown in figure 2. The ease of integration promotes good design practices such as architecture exploration, code reuse, early hardware verification and targeted design components fine-grained refinements. Moreover, current implementation permits simulators to be spreaded across a TCP/IP network allowing distributed processing.

Finally, any simulator or tool can be integrated to the architecture as long as it can be interfaced with a programming language or interfaced with an ORB.

Future work includes evaluating the impact of simulation distribution of a large design over many computers, improving performance and migrating away from MATLAB as a Master. Hardware in the loop verification using an FPGA and the PrismTech's OpenFusion Integrated Circuit ORB is also considered.

Acknowledgement

We acknowledge the support of the Canadian Microelectronics Corporation (CMC) and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] J. Bergeron, "Writing Testbenches using SystemVerilog," *Springer*, 2006.

[2] S. Schmerler, Y. Tanurhan, and K. Muller-Glaser, "A backplane approach for cosimulation in high-level system specification environments," *eurodac*, vol. 00, p. 262, 1995.

[3] K. Muller-Glaser, G. Frick, E. Sax, and M. Kuhl, "Multiparadigm modeling in embedded systems design," *Control Systems Technology, IEEE Transactions on*, vol. 12, no. 2, pp. 279–292, 2004.

[4] H. Patel, D. Mathaikutty, D. Berner, and S. Shukla, "CARH: service-oriented architecture for validating system-level designs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 8, pp. 1458–1474, Aug. 2006.

[5] S. Neema, G. Karsai, and A. Lang, "Tool integration patterns," *Proceedings of Workshop on Tool Integration in System Development, European Software Engineering Conference*, pp. 33–38, 2003.