A Cosimulation Framework for a Distributed System of Systems

Bernd Müller-Rathgeber
Institute of Communication Networks
Technische Universität München
Email: mueller-rathgeber@tum.de

Holm Rauchfuss
Institute for Integrated Systems
Technische Universität München
Email: holm.rauchfuss@tum.de

***Abstract*—In this paper, we present a simple but powerful solution to combine two different simulation environments - SystemC and OMNeT++ - enabling a cosimulation framework for modeling and simulating a distributed system of systems. It is therefore possible to utilize the strengths and preliminary work of OMNeT++ in the field of communication networks and SystemC in modeling hardware entities. We use a socket solution based on standard RPC mechanisms with the possibility to connect the simulation environments over the Internet. Besides the cosimulation interface itself, we show performance and resource utilization measurements of this solution.**

I. Motivation

The purpose of this work was to create the possibility for hardware architecture exploration in an automotive environment. The focus is on an early state of the product-development process, were decisions on used technologies are made with only an abstract specification of the needed resources and functions.

State of the art in the automotive industries are predominantly physical layer simulations, i.e. bit-wise accurate reproductions of parts of the network ("Rest Bus") or electronic control units, respectively sensors and actuators ("Hardware in the Loop"). They are carried out with product samples to verify the specified characteristics. Due to the need of real working hardware, this approach can only take place in a late phase, in which elementary (hardware) design changes are unfeasible or highly expensive. Besides this, the fast increase of complexity of architectures requires a solution - besides mathematical techniques - to make an estimation of the performance and benchmarks of different implementation possibilities as early as possible.

The work of Fummi et al. [1], [2], [3], [4] inspired us to combine the popular simulation environment for communication networks, OMNeT++ (OMNeTpp) (see Chapter II-B), with the simulation environment for hardware entities, SystemC (see Chapter II-A), into one combined framework.

With our framework, it is possible to model and simulate a distributed system of systems, i.e. interconnected embedded systems, with both the embedded systems and the communication network in detail and do performance measurements and assessments on them. Although our focus is on novel automotive IT architectures, the framework is usable for any such kind of system.

A. The Framework

In our approach, the distributed system of systems is divided into 3 parts, representing 3 different steps when processing events (see Figure 1). First, a stimuli-model is executed generating an event at a simulation time t_0. We use intelligent models as stimuli that generating events and corresponding data to be processed, representing encoded video- and audio-streams, raw data of lidar and radar sensors as well as generic sensor input. The second step is the processing of the data in a hardware device causing a delay Δt_1. This delay calculation is done in SystemC. After that, the processed data is transferred over a communication network to another processing unit or an actuator resulting in a delay Δt_2 through the network, based on the model of the automotive bus systems in OMNeTpp.

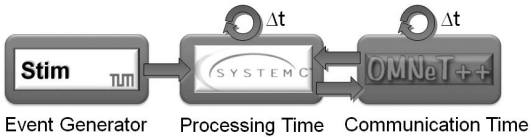

Fig. 1. Simulation Framework

By this approach we do not observe incoherent events in different hardware units, but rather a complete flow of a function-chain existing in modern embedded systems. For example, a signal is generated on a dedicated sensor node, pre-processed there and transmitted via communication network to a control unit where the corresponding calculations are performed and the result is forwarded again via a communication system to the final actor node. As result and for statistic collection, we now have the whole processing delay and therefore the system response time or information on deadline violations.

B. Why two of them?

By combining the two simulation environments it is possible to leverage them in several aspects:

- **Domain-specific modeling:** With the option to model the respective system model part - either an embedded system or communication network - with the most fitting simulation environment allows a good mapping of models without deforming the environment or the understanding of it.
- **Incorporate previous work:** Previous works done in both domains, e.g. adopting an existing communication network from OMNeTpp or an embedded system model in System is possible.
- **Composability:** Selecting two simulation environments with defined interfaces enables to replace the underlying

978-1-4244-1722-3/08/$25.00 ©2008 IEEE.

models with ease, for example replacing Ethernet with Flexray as a communication network.
- **Community:** Both simulation environments have extensive developer communities - both can be addresses with this combined simulator.

The following paper is structured as follows: In Chapter II an introduction to the two underlying simulation environments is given. Chapter III describes the cosimulation interface and its parts, whose performance is analyzed in Chapter IV. The paper concludes with a discussion and outlook in Chapter V.

II. Simulation Environments

In the next sub-chapters an introduction to the two used simulation environment, SystemC and OMNeTpp, is given.

A. SystemC

SystemC [5] is a environment for system-level modeling, design and verification. It extends C++ in form of a library with an event-based simulation kernel, macros, basic data types, constructs and functions for modeling and simulating (concurrent) hardware and software components alike. On top of these basic blocks further extension are available, i.e. transaction-level master/slave interfaces and verification standards.

The SystemC language is approved by the the Institute of Electrical and Electronics Engineers (IEEE) as IEEE standard 1666™-2005 and an open-source reference implementation is provided by the "Open SystemC Initiative" (OSCI) free of charge.

Choosing C++ as underlying programming language instead of traditional hardware description languages, e.g. VHDL or Verilog, allows to capitalize on the great number of developers and the flexibility and expressiveness of this language, but with the cost of currently incomplete conversion to a hardware-synthesizeable format.

SystemC is widely used in public research, but also in industrial products for modeling systems from the high-level architecture description down to the Register Transfer Level (RTL).

For our simulation, functions are modeled only on an abstract level on which no real functionality or implementation is required. Resource accesses are represented on transaction-level using write and read requests with their respective data amount. Internal working of functions are dynamic-calculated time delays. Embedded systems are modeled with their basic hardware blocks, e.g. cores, memories or buses, and software blocks, e.g. operation system or process scheduler. Every block is represented as element with basic functionality so that allocation and resource utilization can be simulated with the abstract functions mentioned above.

B. OMNeTpp

OMNeTpp [6] is a modular, open-source discrete simulation environment. The major features are:

- Written in C++ and easily expandable with dynamic libraries.
- Own topology description language *NED* with XML converter.
- Several open-source simulation models available, for example the *INET*-framework as base of our Ethernet simulations.
- Exchangeable control environment: GUI for runtime control or debugging and command line for fast and automatic cycles.

Its primary application area is the simulation of communication networks, but the type of application is growing rapidly.

OMNeTpp has an event-based simulation kernel. The events are named messages and are stored in the *MessageHeap* queue for eradication, and so trigger modules that them self in turn generate new messages.

For our simulation, the communication network including Medium Access Control (MAC) itself is modeled on packet-level-granularity and with the respective processing and transmission time calculation.

III. The Cosimulation Interface

The approach of combining two simulations for a complex cosimulation is not new. Most solutions were developed in the field of HW/SW-cosimulation [7], [8]. Distinguishing features are performance of the framework and timing accuracy. For the synchronization of the two separate event kernels, several techniques were discussed. One solution is using *rollback* [9], so going back in time when one simulator receives a past event from the other simulator. Others synchronize the clocks when exchanging events or in a periodically manner [10], [11], [12]. Both of our simulation environments are discrete event-based [13], [14] and can only step forward in the simulation time. So *rollback* is no alternative and we have to especially take care not generating *"Events in Past"*-errors crashing the whole simulation.

In our solution, the two simulation kernels are only lightly coupled and communicate socket-based with each other exchanging timing information and simulation data. Both retain their own local simulation time and event queue.

A. Communication

For communication between these two simulation kernels, we take a network-based approach, allowing to run the two simulation environments on different machines in a network. Without having to implement an own socket interface, we use the Open Network Computing Remote Procedure Call (ONC RPC) [15] of Linux. The implementation in SystemC and OMNeTpp is based on the RPC++ code by Ramon E. Creager from the National Radio Astronomy Observatory [16].

In SystemC, the RPC++ code is centralized in one SystemC module (called *Syncinterface*, see Figure 2). It is abstracting the communication network for the via ports/channels connected hardware entities. The *Syncinterface* has the role to control OMNeTpp (see Chapter III-B) and route messages between OMNeTpp and the simulated hardware entities (see Chapter III-C). On the OMNeTpp side, all code modification are concentrated in a modified version of the shipped

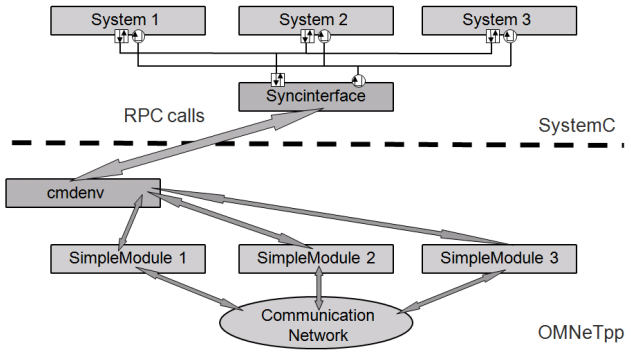

Fig. 2. RPC Interface

command line environment library (*cmdenv*), not touching the core libraries of OMNeTpp at all. The changes were mainly wrapper classes for the RPC procedures and an event loop for the RPC server. The *SimpleModules* are stubs representing the hardware entities in OMNeTpp.

In a RPC setup, the server (OMNeTpp in our case) registers the available remote procedures addressed by a system wide unique ID at a directory service, the Linux *Portmapper*-daemon. The client (SystemC) sends a message with the ID and the needed parameters to the server, starting the remote process and waiting for the return parameters. We use *synchronous RPC*, so the client must wait for an answer of the server. This helps to guarantee a strict sequential simulation flow. In Chapter IV, we present some performance measurements and discussions about this protocol.

The available remote procedures are:
name(send information, return information): description

- *getnextEvent*(NULL, simulation time):
  queries OMNeTpp *MessageHeap*.
- *runUntil*(simulation time, status):
  sets a breakpoint in OMNeTpp.
- *sendMessage*(message, status):
  generates new event **in** OMNeTpp.
- *readMessage*(NULL, message):
  receives event **from** OMNeTpp.
- *control*(command, status):
  for starting, stopping, selecting run, ...

getnextEvent and *runUntil* are described in Chapter III-B, *sendMessage* and *readMessage* in Chapter III-C.

B. Synchronization

Time alignment of two simulation environments can be realized according to two conceptional schemes. *Asymmetric* means one simulation kernel controls the execution of the other, or *symmetric* where both environments work separated and time synchronization is achieved by exchanging messages. Contrary to [3] our approach is *asymmetric* resulting in a less complex synchronization implementation. In our framework, SystemC acts as master and OMNeTpp takes the role of the sole slave. This decision is feasible because no message is initially created in OMNeTpp.

In a complex hardware scenario like simulated in our framework and representative for automotive applications, a function chain starts with an external trigger representing an event that generates a sensor input. From this sensor, an information (or message) is created and traverses the system and leading to an actuator output, terminating this message. Sensor and actuator are simulated in SystemC, so every message starts and ends there. It is therefore OMNeTpp can be remote-controlled by SystemC. This does not prohibit OMNeTpp to generate messages for other entities simulated inside the OMNeTpp environment, for example configuration or control messages for the switches. Only the transfer to a SystemC modeled host is not possible by design.

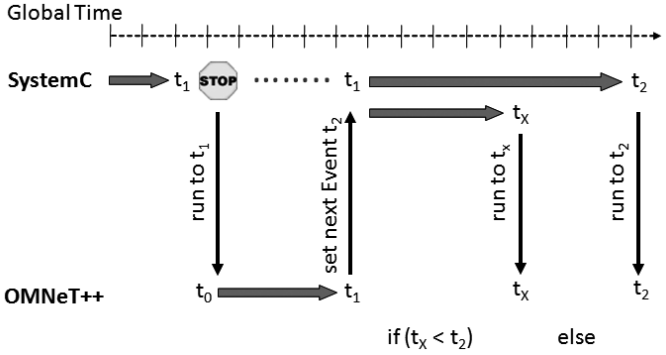

Fig. 3. Synchronization

Figure 3 illustrates the flowchart of the synchronization. SystemC starts when an external trigger from the stimuli-model occurs.

Two new (remote) procedures control OMNeTpp. *runUntil*(t) starts the execution of the message queue and proceeds until the first (the one with the smallest timestamp) message of the *MessageHeap* has a timestamp past t. This is done by performing single steps and subsequent reading the message queue. Like mentioned above, this procedure locks the progress in SystemC until the last message is processed and OMNeTpp enters the event loop waiting for further RPC calls. *getnextEvent*(t) returns the timestamp of the first message in the *MessageHeap*. This value it transmitted to SystemC with the return parameter of the RPC call preventing SystemC from missing events in OMNeTpp.

This means for our solution:

1) At any time, only **one** of the two simulation environments is active and processes his event queue.
2) SystemC starts and OMNeTpp follows in the sense of the simulation time.
3) Context switches only occur when a message/event has to be processed, not periodically.

We call this a *stop-and-go* synchronization interface.

C. Data Exchange

The other implemented remote procedures *sendMessage()* and *readMessage()* are for exchanging data/messages between

the two environments, i.e. generating events in SystemC and OMNeTpp.

A message is a C++ class that consist of two parts. The parent class is used in the synchronization interface and is independent from the simulated hardware respectively network. It contains the source module in OMNeTpp, where the associated message will be generated and the simulation time, when this should happen. The second part of the class is derived from the parent and contains all necessary fields needed for the simulated system, for example message size and priority.

When one embedded system in SystemC wants to exchange data over a network with another embedded system, it instantiates the class and set the necessary fields *source, target, send time* and the simulation dependent fields *message size, priority, ...* The serializing call copies the data in a *u_char*-Array serving as buffer. The buffer is than transmitted as a *(xdrproc_t) xdr_wrapstring* parameter in the *sendMessage()* RPC call to OMNeTpp. In OMNeTpp, a new and empty generic event will be scheduled at the stated simulation time for the source module.

On reception, the target module updates the fields of the class. At every *stop-and-go*, OMNeTpp browses the *Message-Heap* for the appearance of such a message and enters an idle state until SystemC polls the message.

IV. Performance

Performance measurements are difficult to set up, because there is always a problem with comparability. The speed of a simulation primarily depends on the simulated model and used hardware. We utilize an Intel Core Duo with two cores of $2.13GHz$ processor clock and $2GB$ of RAM for our framework running on Linux. In this chapter, we give an overview of the achieved simulation speed and some generic, hence transferable performance indexes of our cosimulation interface.

A. Simulation Speed

Our system consists of fourteen embedded systems connected to six different Ethernet switches building the core network (see Figure 4). Spanning several dozen function-chains the embedded systems generate messages every $128\mu s$ in average and send them to a different node.

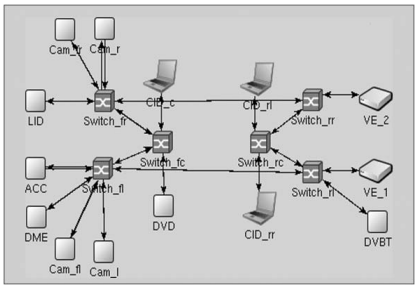

Fig. 4. Sample Network

Table I shows the most important results.

5	seconds simulation time
665	seconds wall clock time
7754905	Events in OMNeTpp
548172	Messages transferred
3592251	Number of *stop-and-go* of OMNeTpp
2.2	Events per *stop-and-go* of OMNeTpp
$1391ns$	simulation time per *stop-and-go* of OMNeTpp

TABLE I
Simulation Speed

We achieved a total speed of $133\,Sec/Simsec$ by $\approx 1,55 \cdot 10^6\,Ev/Sec$. Also the *stop-and-go* synchronization shows his advantage: At a time resolution of nanoseconds, the system switches only every 1391 ticks ($1391ns$) on average.

In the next Chapters, we take a closer look to the CPU performance (Chapter IV-B) and the data traffic on the socket-based cosimulation interface (Chapter IV-C).

B. Processing Time

For profiling the CPU time of the simulation, OProfile [17] has been used - a statistical profiling tool on function call level.

Of the total simulation time 31% (206.01 seconds) are used up by SystemC, 69% (459.29 seconds) by OMNeTpp, shown in Figure 5.

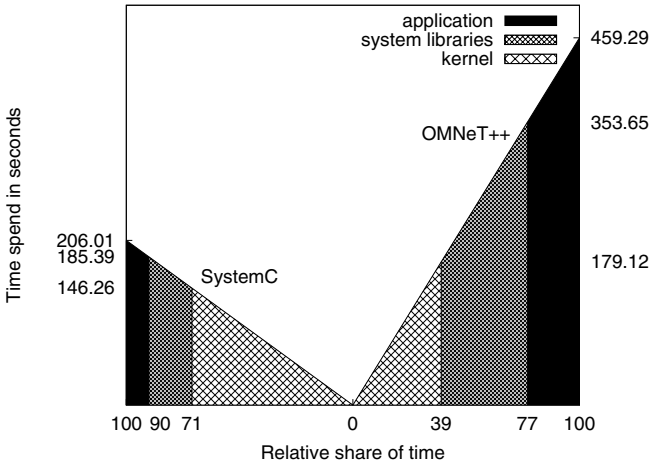

Fig. 5. Time spend in SystemC and OMNeTpp regarding application, system libraries and kernel

C. Socket

With the help of tcpdump [18], we recorded the packets on the cosimulation interface and analyzed them.

Figure 6 displays the charts for down- and upstream by measuring the filtered TCP requests. With an average packet size of 115 $Byte$ per request and 102 $Byte$ per response we get a downstream (SystemC $\rightarrow$ OMNeTpp) of 17.0 $MBit/s$ and an upstream (OMNeTpp $\rightarrow$ SystemC) of 15.1 $MBit/s$.

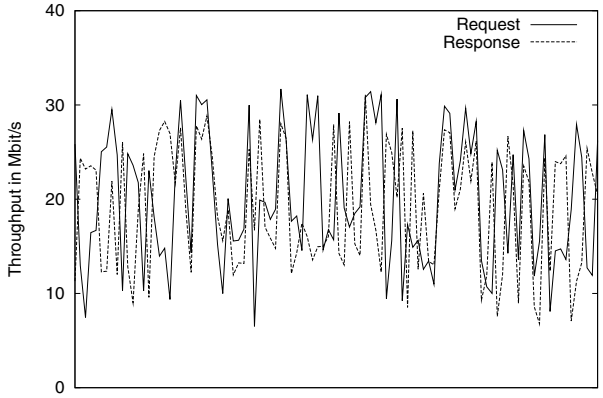

Fig. 6. TCP throughput

V. Discussion and Outlook

The implementation of the cosimulation interface with the help of RPC was easy to achieve. With the blocking RPC calls and the event loop of the RPC server we avoid the use of parallel threads and synchronization between them. Also the exchange of data is possible with the shipped procedures. Through the socket-based interface, we had a simple possibility to debug the activity chain by recording and interpreting the TCP packets. Besides that, it is possible to separate the two simulation environments on two over the Internet connected servers on different locations or include another environment, e.g. an event generator. So the implementation of the models as well as the work on OMNeTpp and SystemC could take place at different institutes.

We used an average-sized scenario, data and events, to show that the framework with the cosimulation interface as well as the *stop-and-go* mechanism works correctly with an acceptable performance.

A. Future Work

A lot of time is "wasted" in kernel space processing RPC messages. Additional measurements not being the focus of this paper showed, that the number not the size of the packets is deciding. So the priority task is to decrease the needed RPC calls to 1 per *stop-and-go* merging the result of *getnextEvent()* and the state of pending messages into the return parameter of the *runUntil()* call.

B. Conclusion

Our synchronization solution is a small and easy to implement method for connecting two simulation environments. In OMNeTpp, only one dynamic library has to be changed with about 160 Lines of Code (LOC) for the RPC server and 100 LOC for the message handling plus adding two existing independent static libraries. In SystemC, the appropriate changes are about 200 LOC in the model code for adding the RPC client interface. There was no need to touch the event simulation core library at all. We use RPC as a standardized protocol for client-server systems included in all major Linux distributions.

VI. Organization

This work is part of an interdisciplinary research project of the Technische Universität München and BMW Group Research and Technology. It is organized under the CAR@TUM initiative, where the decades-worth cooperation between the Technische Universität München and the BMW Group is intensified and restructured. The field "*IT-Architecture*" is one of six initiated high-tech projects and has its scope on new approaches and design rules for an automotive Electrics and Electronics (E/E) architecture.

References

[1] N. Drago, F. Fummi, and M. Poncino, "Modeling network embedded systems with ns-2 and systemc," in *1st IEEE International Conference on Circuits and Systems for Communications, ICCSC'02.*, 2002, pp. 240–245.

[2] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Legacy systemc co-simulation of multi-processor systems-on-chip," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2002.*, 2002, pp. 494–499.

[3] F. Fummi, G. Perbellini, P. Gallo, M. Poncino, S. Martini, and F. Ricciato, "A Timing-Accurate Modeling and Simulation Environment for Networked Embedded Systems," *Proceedings of the 40th Design Automation Conference (DAC03)*, vol. 1, pp. 58 113–688, 2003.

[4] F. Fummi, M. Loghi, S. Martini, M. Monguzzi, G. Perbellini, and M. Poncino, "Virtual Hardware Prototyping through Timed Hardware-Software Co-Simulation," *Proceedings of the Design, Automation and Test in Europe (DATE'05) Volume 2-Volume 02*, pp. 798–803, 2005.

[5] OSCI, "SystemC, a system-level modelling, design and verification language," *Homepage: http://www. systemc. org*.

[6] A. Varga *et al.*, "The OMNeT++ discrete event simulation system," *Proceedings of the European Simulation Multiconference (ESM2001)*, 2001. [Online]. Available: http://www.omnetpp.org/

[7] P. Coste, F. Hessel, P. Le Marrec, Z. Sugar, M. Romdhani, R. Suescun, N. Zergainoh, and A. Jerraya, "Multilanguage design of heterogeneous systems," *Hardware/Software Codesign, 1999.(CODES'99) Proceedings of the Seventh International Workshop on*, pp. 54–58, 1999.

[8] J. Buck, S. Ha, E. Lee, and D. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal of Computer Simulation*, vol. 4, no. 2, pp. 155–182, 1994.

[9] S. Yoo and K. Choi, "Optimistic Timed HW-SW Cosimulation," *Proc. 4th Asia-Pacific Conference on Hardware Description Language*, pp. 39–42, 1997.

[10] W. Sung and S. Ha, "Optimized timed hardware software cosimulation without roll-back," *Proceedings of the conference on Design, automation and test in Europe*, pp. 945–946, 1998.

[11] D. Kim, C. Rhee, Y. Yi, S. Kim, H. Jung, and S. Ha, "Virtual synchronization for fast distributed cosimulation of dataflow task graphs," *System Synthesis, 2002. 15th International Symposium on*, pp. 174–179, 2002.

[12] Y. Yi, D. Kim, and S. Ha, "Virtual synchronization technique with OS modeling for fast and time-accurate cosimulation," *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 1–6, 2003.

[13] R. Righter and J. Walrand, "Distributed simulation of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 99–113, 1989.

[14] P. Ball, "Introduction to discrete event simulation," *2nd DYCOMANS workshop on Management and Control: Tools in Action*, pp. 367–376, 1996. [Online]. Available: http://www.dmem.strath.ac.uk/ pball/simulation/simulate.html

[15] R. Srinivasan, "Binding Protocols for ONC RPC Version 2," *Request for Comments (Proposed Standard)*, 1833.

[16] R. Creager, "ZY Version 2.5 Software Interface Reference Manual," *National Radio Astronomy Observatory, PO Box*, vol. 2, pp. 24 944–0002. [Online]. Available: http://www.gb.nrao.edu/GBT/MC/ygor/libraries/RPC++/

[17] J. Levon *et al.*, "OProfile, a system-wide profiler for Linux systems," *Homepage: http://oprofile. sourceforge. net*.

[18] V. Jacobson, C. Leres, S. McCanne *et al.*, "Tcpdump," *available via anonymous ftp to ftp. ee. lbl. gov*, 1989.