A SystemC-based Framework for Modeling and Simulation of Networked Embedded Systems

F. Fummi, D. Quaglia, F. Stefanni
Dipartimento di Informatica, Università di Verona, strada le Grazie 15, I-37134, Verona, Italy
[franco.fummi|davide.quaglia|francesco.stefanni]@univr.it

Abstract

Next-generation networked embedded systems pose new challenges in the design and simulation domains. System design choices may affect the network behavior and Network design choices may impact on the System design. For this reason, it is important—at the early stages of the design flow—to model and simulate not only the system under design, but also the heterogeneous networked environment in which it operates. For this purpose, we have exploited a modeling language traditionally used for System design—SystemC—to build a System/Network simulator named SystemC Network Simulation Library (SCNSL). This library allows to model network scenarios in which different kinds of nodes, or nodes described at different abstraction levels, interact together. The use of SystemC as unique tool has the advantage that HW, SW, and network can be jointly designed, validated and refined. As a case study, the proposed tool has been used to simulate a sensor network application and it has been compared with NS-2, a well-known network simulator; SCNSL shows nearly two-order-magnitude speed up with TLM modeling and about the same performance as NS-2 with a mixed TLM/RTL scenario. The simulator is partially available to the community at http://sourceforge.net/projects/scnsl/.

1 Introduction

The widespread use of Networked Embedded Systems (i.e., embedded systems with communication capabilities like PDAs, cellphones, routers, wireless sensors and actuators) has generated significant research for their efficient design and integration into heterogeneous networks [5, 7–9, 11, 13, 14].

The design of such networked embedded systems is strictly connected with network functionality as suggested in the literature [3]. In fact, choices taken during the System design exploration may influence the network configuration and viceversa. The result is a two-dimension design space as depicted in Figure 1. System and Network design can be seen as two different aspects of the same design problem, but they are generally addressed by different people belonging to different knowledge domains and using different tools. In the context of System design, tools aim at providing languages to describe models and engines for their simulation. The focus is the functional description of computational blocks and the structural interconnection among them. Particular attention is given to the description of concurrent processes with the specification of the synchronization among them through wait/notify mechanisms. The most popular languages in this context are VHDL, Verilog, System Verilog, and SystemC [1]; in particular, the last language is gaining acceptance for its flexibility in describing both HW and SW components and for the presence of add-on libraries for Transaction-Level Modeling (TLM) and verification. In the context of network simulation, current tools reproduce the functional behavior of protocols, manage time information about transmission and reception of events and simulate packet losses and bit errors. Network can be modeled at different levels of detail, from packet transmission down to signal propagation [2, 4, 12].
The use of a single tool for System and Network modeling would be an advantage for the design of networked embedded systems. Network tools cannot be used for System design since they do not model concurrency within each network node and do not provide a direct path to hardware/software synthesis. Instead, System tools might be the right candidate for this purpose since they already model communications, at least at system level. However, the use of a system description language for network modeling requires the creation of a basic set of primitives and protocols to support the asynchronous transmission of variable-length packets. To fill this gap, in this work we have evaluated the potentiality of SystemC in the simulation of packet-switched networks. In the past, SystemC was successfully used to describe network-on-chip architectures [6] and to simulate the lowest network layers of the Bluetooth communication standard [5]. In the proposed simulator, devices are modeled in SystemC and their instances are connected to a module that reproduces the behavior of the communication channel; propagation delay, interference, collisions and path loss are taken into account by considering the spatial position of nodes and their on-going transmissions. The design of the node can be dealt at different abstraction levels: from system/behavioral level (e.g., Transaction Level Modeling) down to register transfer level ( RTL) and gate level. After each refinement step, nodes can be tested in their network environment to verify that communication constraints are met. Nodes with different functionality or described at different abstraction levels can be mixed in the simulation thus allowing the exploration of network scenarios made of heterogeneous devices. Synthesis can be directly performed on those models provided that they are described by using a suitable subset of the SystemC syntax.

The paper is organized as follows. Section 2 describes the SystemC Network Simulation Library. Section 3 outlines the main solutions brought by this work. Section 4 reports experimental results and, finally, conclusions are drawn in Section 5.

2 SCNSL Architecture

The driving motivation at the base of SCNSL is to have a single simulation tool to model both the embedded system under design and the surrounding network environment. SystemC has been chosen for its great flexibility, but a lot of work has been done to introduce some important elements for network simulation. SCNSL takes in charge the translation of network primitives (e.g., packets events) into SystemC primitives.

2.1 Main components

To support network modeling and simulation, a tool has to provide the following elements:

- **Kernel**: the kernel is responsible for the correct simulation, i.e., its adherence to the behavior of an actual communication channel; the kernel has to execute events in the correct temporal order and it has to take into account the physical features of the channel such as, for example, propagation delay, signal loss and so forth.

Figure 2. SCNSL in the context of SystemC modeling.
Figure 3. Main components of SCNSL.

- **Node**: nodes are the active elements of the network; they produce, transform and consume transmitted data;
- **Packet**: in packet-switched networks the packet is the unit of data exchanged among nodes; it consists of a header and a payload;
- **Channel**: the channel is an abstraction of the transmitting medium which connects two or more nodes; it can be either a point-to-point link or a shared medium;
- **Port**: nodes use ports to send and receive packets.

Figure 3 shows the main components of SCNSL; they can be easily related to the previous list as explained below. The *simulation kernel* is implemented by the Network_if_t class. This class is the most complex object of SCNSL, because it manages transmissions and, for this reason, it must be highly optimized. For instance, in the wireless model, the network transmits packets and simulates their transmission delay; it can delete ongoing transmissions, change node position, check which nodes are able to receive a packet, and verify if a received packet has been corrupted due to collisions. The standard SystemC kernel does not address these aspects directly, but it provides important primitives such as concurrency models and events. The network class uses these SystemC primitives to reproduce transmission behavior. In particular, it is worth to note that SCNSL does not have its own scheduler since it exploits the SystemC scheduler by mapping network events on standard SystemC events.

The *Node* is one critical point of our library which supports both System and Network design. From the point of view of a network simulator the node is just the producer or consumer of packets and therefore its implementation is not important. However, for the system designer, node implementation is crucial and many operations are connected to its modeling, i.e., change of abstraction level, validation, fault injection, HW/SW partitioning, mapping to an available platform, synthesis, and so forth. For this reason we introduced the class NodeProxy_if_t which decouples node implementation from network simulation. Each Node instance is connected to a NodeProxy instance and, from the perspective of the network simulation kernel, the NodeProxy instance is the alter-ego of the node. This solution allows to keep a stable and well-defined interface between the NodeProxy and the simulation kernel and, at the same time, to let complete freedom in the modeling choices for the node; as depicted in Figure 3 the box named Node is separated from the simulation kernel by the box named NodeProxy and different strategies can be adopted for the modeling of the node, e.g., interconnection of basic blocks or finite-state machine. It is worth to note that other SystemC libraries can also be used in node implementation, e.g., re-used IP blocks and testing components such as the well-known SystemC Verification Library. For example, the Figure also shows an optional package above the node; this package is provided by SCNSL and it contains some additional SystemC modules, i.e., an RTL description of a timer and a source of stimuli. These components may simplify designer’s work even if they are outside the scope of network simulation.

Another critical point in the design of the tool has been the concept of *packet*. Generally, packet format depends on the corresponding protocol even if some features are always present, e.g., the length and source/destination address. System design requires a bit-accurate description of packet contents to test parsing functionality while from the point of view of the network simulator the strictly required fields are the length for bitrate computation and some flags to mark collisions (if routing is performed by the simulator, source/destination addresses are used too). Furthermore, the smaller the number of different packet formats, the more efficient is the simulator implementation. To meet these opposite requirements in SCNSL, an internal packet format is used by the simulator while the system designer can use other different packet formats according to protocol design. The conversion between the user packet format and the internal packet format is performed in the NodeProxy.

*Channels* are very important components, because they are an abstraction of the transmission media. Standard SystemC channels are generally used to model interconnections between HW components and, therefore, they can be used to model network at physical level [5]. However, many general purpose network simulators reproduce transmissions at packet level to speed up simulations. SCNSL follows this approach and provides a flexible channel abstraction named Communicator_if_t. A communicator is the most abstract transmission component and, in fact, both NodeProxy and Network classes derive from it. New capabilities and behavior can be easily added by extending this class. Communicators can be interconnected each other to create chains. Each valid chain shall have on one end a
NodeProxy instance and, on the other end, the Network; hence transmitted packets will move from the source NodeProxy to the Network traversing zero or more intermediate communicators and then they will eventually traverse the communicators placed between the Network and the destination NodeProxy. In this way, it is possible to modify the simulation behavior by just creating a new communicator and placing its instance between the network and the desired NodeProxy.

Figure 4 shows the class hierarchy of the Communicator; as said before, both Network and NodeProxy inherit from the Communicator. A wireless network is a specific kind of Network with its own behavior and thus derives from the abstract Network. A possible implementation of a wireless network model is described in Section 3.4. NodeProxies depend both on the type of network and on the abstraction level used in node implementation; for example, Figure 4 reports a TLM and an RTL version of a wireless NodeProxy.

3 SCNSL main resolved problems

This Section describes some issues encountered during the development of SCNSL and the adopted solutions. The first problem regards the co-existence of RTL system models with the packet-level simulation. The second one regards the assessment of packet validity with reference to collision and out-of-range transmissions. The third problem regards the planning of the network simulation, i.e., source activations, link failures, and so forth. Finally, an application of SCNSL to a wireless network is described.

3.1 Simulation of RTL models

As said before, SCNSL supports the modeling of nodes at different abstraction levels. In case of RTL models, the co-existence of RTL events with network events has to be addressed.RTL events model the setup of logical values on ports and signals and they have an instantaneous propagation, i.e., they are triggered at the same simulation time in which the corresponding values are changing. Furthermore, except for tri-state logic, ports and signals have always a value associated to them, i.e., sentences like “nothing is on the port” are meaningless. Instead, network events are mainly related to the transmission of packets; each transmission is not instantaneous because of transmission delay, during idle periods the channel is empty, and the repeated transmission of the same packet is possible and leads to distinct network events.

In SCNSL, RTL node models handle packet-level events by using three ports signaling the start and the end of each packet transmission, and the reception of a new packet, respectively. Also in this case the NodeProxy instance associated to each node translates network events into RTL events and vice versa. In particular, each RTL node has to write on a specific port when it starts the transmission of a packet while another port is written by the corresponding NodeProxy when the transmission of the packet is completed. A third port is used to notify the node about the arrival of a new packet. With this approach each transmission/reception of a packet is detected even if packets are equal.

The last issue regards the handling of packets of different size. Real world protocols use packets of different sizes while RTL ports must have a constant width set at compile-time. SCNSL solves this problem by creating packet ports with the maximum packet size allowed in the network scenario and by using an integer port to communicate the actual packet size. In this way a NodeProxy or a receiver node can read only the actual used bytes, thus obtaining a correct simulation.

3.2 Transmission validity assessment

In wireless scenarios an important task of the network simulator kernel is the assessment of transmission validity which could be compromised by collisions and out-of-range distances. The validity check has been implemented by using two flags and a counter. The first flag is associated to each node pair and it is used to check the validity of the transmission as far as the distance is concerned; if the sender or the receiver of an ongoing transmission has been moved outside the maximum transmission range, this flag is set to false. The second flag and the counter are associated to each node and they are used to check the validity with respect to collisions. The counter is used to register the numbers of active transmitters which are interfering at a given receiver; if the value of this counter is greater than one, then on-going transmission to the given receiver are not valid since they are compromised by collisions. However, even if, at a given time, the counter holds one, the transmission could be invalid due to previous collisions; the flag has the purpose to track this case. When a packet transmission is completed, if the value of the counter is greater than one, the flag is set to false. The combined use of the flag and
the counter allows to cover all transmission cases in which packet validity is compromised by collisions.

### 3.3 Simulation planning

In several network simulators, special events can be scheduled during the setup of the scenario; such special events regard nodes movements, link status changes, traffic activation and packet drops. This feature is important because it allows to simulate the model into a dynamic network context. In SCNSL the simulation kernel has not its own event dispatcher, hence this feature has been implemented into an optional class, called EventsQueue_t. Even if SystemC allows to write in each node the code which trigger such events, the choice of managing them in a specific class of the simulator leads to the following advantages:

- **Standard API**: the event queue provides a clear interface to schedule a network event without directly interacting with the Network class or altering node implementation.
- **Simplified user code**: Network events are more complex then System ones; the event queue hides such complexity thus simplifying user code and avoiding setup errors.
- **Higher performance**: the management of all the events inside a single class improves performance; in fact the event queue is built around a single SystemC thread, minimizing memory usage and context switching.

This class can be used also to trigger new events defined through user-defined functions. The only constraint is that such functions shall not block the caller, i.e., the events queue, to allow a correct scheduling of the following events.

### 3.4 Application to a wireless scenario

In this Section the behavior of the library is described with reference to a wireless scenario; in particular, an RTL node model is reported to clarify the concepts written in Section 3.1. Module Rtl::Node_t represents an abstract network node. It has a set of properties which are used by the simulation framework to reproduce network behavior. Transmission rate represents the number of bits per unit of time which the interface can handle; it is used to compute the transmission delay and the network load. Transmission power is used to evaluate the transmission range and the signal-to-noise ratio. Transmission rate and transmission power can be changed during simulation to accurately simulate and evaluate power saving algorithms. The module has the following ports:

- **packet ports** to send and receive packets, respectively;
- **carrier port** to perform carrier sense;
- **packet length ports** to report actual packet size (one for each direction);
- **packet event management ports** to report the presence of a new packet (one for each direction) and the completion of packet transmission;
- **rate port** to communicate the transmission rate to the simulation kernel through the NodeProxy;
- **power port** to communicate the transmission power to the simulation kernel through the NodeProxy;
- **sensor port** to model a data input whose meaning is application-specific (e.g., a temperature sensor).

The sensor port of each node is bound to an instance of the module Stimulus_t which reproduces a generic environmental data source. It takes as input a clock signal as timing reference to synchronize the generation of data values. Different kinds of stimuli can be generated by subclasses of this module; the intensity of the stimuli and their localization in time can follow a given statistical distribution or be derived from a trace file.

The class Rtl::NodeProxy_t interfaces the node with the network and it manages two node properties, i.e., node position and receiver sensitivity. Node position is used to compute the path loss and to reproduce mobile scenarios. The receiver sensitivity is the minimum signal power below which the packet cannot be received. Even if these properties are related to the node they are frequently used by the simulation kernel and thus we decided to model them in the NodeProxy to simplify their access.

When a node starts transmission, its relative position to all other nodes in the same network is computed, and the signal level in all those nodes is derived according to the path loss formula $1/d^\alpha$. For each node, if the signal level is higher than its receiver sensitivity, then it can be detected and it may interfere with other on-going transmissions. If there are already on-going transmissions reaching the receiving node, then all those messages are marked as collided (i.e., they are not valid). Also, if there are other on-going transmissions which the currently sending node reaches with its transmission, then those messages are marked as collided as well. Since wireless nodes cannot detect collisions, a collided message is not interrupted and the channel remains busy. The transmission time depends on the packet length, the transmission rate, and the propagation delay.

### 4 Experimental results

The SystemC Network Simulation Library has been used to model a wireless sensor network application consisting of a master node which repeatedly polls sensor nodes to obtain data. Node communications reproduce a subset of the well-known IEEE 802.15.4 standard, i.e., peer un-slotted transmissions with acknowledge [10].
Different scenarios have been simulated with SCNSL by using nodes at different abstraction levels: 1) all nodes at TLM-PVT level, 2) all nodes at RTL, and 3) master node at RTL and sensor nodes at TLM-PVT. The designer had written 172 code lines for the `main()`, 688 code lines for the RTL node and 633 code lines for the TLM-PVT node.

Figure 5 shows the CPU time as a function of the number of simulated nodes for the different tested tools and node abstraction levels.

The speed of SCNSL simulations at TLM-PVT level is about two-order-magnitude higher than in case of NS-2 simulation showing the validity of SCNSL as a tool for efficient network simulation. Simulations at RT level are clearly slower because each node is implemented as a clocked finite state machine as commonly done to increase model accuracy in System design. However a good trade-off between simulation speed and accuracy can be achieved by mixing nodes at different abstraction levels; in this case, experimental results report about the same performance of NS-2 and about the same accuracy in System design and Network simulation requirements while preserving execution efficiency. In particular, the combined simulation of RTL system models and packet-based networks has been faced. Experimental results for a large network scenario show nearly two-order-magnitude speed up with respect to NS-2 with TLM modeling and about the same performance as NS-2 with a mixed TLM/RTL scenario.

5 Conclusions

We have presented a SystemC-based approach to model and simulate networked embedded systems. As a result, a single tool has been created to model both the embedded system under design and the surrounding network environment. Different issues have been solved to reconcile System

References