An Efficient and Modular Method for the Simulation of Real-Time Wireless Embedded Systems (Poster Abstract)

Jérôme Rousselot
CSEM SA
1 Jaquet-Droz
CH-2002 Neuchâtel
+41 32 720 5829
jerome.rousselot@csem.ch

Jean-Dominique Decotignie
CSEM SA
1 Jaquet-Droz
CH-2002 Neuchâtel
+41 32 720 5000
jean-dominique.decotignie@csem.ch

ABSTRACT
This paper describes a novel method to execute embedded real-time C code within the OMNeT++ network simulator and its wireless extension MiXiM. This method greatly reduces the development and debugging effort of wireless embedded systems, while also significantly improving the accuracy and utility of network simulations. The method is highly modular: it is possible to isolate every layer of the embedded communication protocols stack and combine it with simulation modules, or to combine all of the embedded protocols together. The first mechanism increases the simulation speed and enables the evaluation of novel algorithms in a variety of configurations, while the second mechanism allows debugging the embedded system with a high granularity. Executing the embedded code in the OMNeT++ simulator gives access to a fast simulation kernel with powerful simulation configuration semantics, large simulation model libraries and extensive data collection and analysis tools. This method is now used to develop novel systems in industrial and public projects: public transportation safety, inventory tracking and monitoring of composite structures.

Categories and Subject Descriptors
I.6.5 [Simulation and Modeling]: Model Development – *Modeling methodologies*; I.6.8 [Simulation and Modeling]: Types of simulation - *discrete event*; J.7 [Computer Applications]: Computer in other systems – *real time*

General Terms
Measurement, Performance, Design, Reliability, Verification, Experimentation

Keywords
Wireless sensor networks, OMNeT++, MiXiM, embedded systems.

1. INTRODUCTION
Wireless embedded systems are difficult to debug, because of their limited resources that prevent extensive tracing, and limited physical access due to application specific constraints. In particular, the large size (up to a few hundred) and long deployment durations (often more than a year) of wireless sensor networks (the type of embedded systems considered here) makes detecting all bugs before deployment a challenging task.

Basic system functionality can be tested under the control of a debugger for a few nodes using JTAG interfaces. However, this approach does not scale well, and real-time events cannot be captured by a debugger. Thus network simulators are often used to test the communication protocols. While these tools are powerful, and often comes with extensive model libraries that enable fast evaluation of novel algorithms, developing network simulation models of embedded systems lead to several drawbacks:

- keeping both code bases functionally equivalent is a time-consuming process;
- the validation of this functional equivalence is challenging;.
- higher layer protocols designed and developed in the network simulator must be rewritten almost from scratch in the embedded code base;
- both code bases must be documented;
- due to the added complexity, developers tend to specialize in either the simulation or the embedded code base and associated development environments. This makes the previous problems even more difficult to address.

This work presents a novel method that enables executing the embedded code within the OMNeT++ discrete event simulation engine and its MiXiM wireless networking models library. It allows to freely combine embedded code protocols and simulation models, enabling fast and accurate evaluation of novel protocols. The simulation of the entire stack enables setting up automated regression tests on the source code repository servers, and helps in developing a better understanding of the network's dynamics. Using the OMNeT++ kernel provides access to efficient and reliable data collection and analysis tools, enables fast simulations and offers sophisticated simulation configuration mechanisms.

SIMUTOOLS 2011, March 21-25, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245584

2. RELATED WORK

This section describes related work on tools to facilitate the design and development of communication protocols. We identify three categories: protocol simulation, hardware emulation and platform virtualization. The first category is the most common strategy, in which case two code bases exist and must be maintained. The second category consists of interpreting the firmware used on the sensor nodes, and to connect the emulator to a network simulator. The third category replaces the low-level I/O software modules of the embedded code base.

2.1 Protocol Simulation Models

As testing embedded systems is a time consuming process, network simulators [1, 2] enable developers to evaluate their algorithms in many different situations, by varying the configuration parameters. This allows for instance to estimate the best parameter values for a MAC protocol in a given application scenario. These tools have been developed for years and offer useful features for statistics collection, tracing, debugging, and high execution speed and scalability.

The disadvantage of this approach is that a separate implementation of the protocol is realized in the simulator. Therefore, all the work consists actually of studying the simulation model rather than the embedded system. Any deviation between these two code bases may potentially lead to completely different results. The simulation model developers must regularly update their code to reflect any change to the embedded protocol implementation, and conversely.

2.2 Hardware Emulation

Others [3, 4] have proposed executing the firmware at the assembly level in an emulator representing the embedded system. This offers high accuracy simulations, at the downside however of execution speed, in particular when considering multiple node simulations The emulation system is connected to a network simulator, to allow simulating an entire network. In WorldSens [3], a dedicated simulator was developed, while RaPTEX [4] makes use of the OMNeT++ discrete event simulator. Both tools suffer from slow execution speed. They rely on complex mechanisms to overcome this limitation, such as speculative execution or network partitioning. RaPTEX also introduces analytical models to avoid the simulation speed bottleneck.

As they simulate the network and emulate the hosts, what happens at the host level cannot be recorded by the network simulator, and network simulation models can only be used for mobility and propagation (however, network partitioning perturbs it).

2.3 Platform Virtualization

TOSSIM [5] is a simulator designed exclusively for TinyOS [6] based systems, a programming framework that facilitates embedded development. In TOSSIM, the low-level hardware interfacing modules are redefined. This increases the execution speed compared to hardware emulation, but requires developing a dedicated network simulator. The TOSSIM simulator does not model energy consumption, and offers basic propagation and interference models.

2.4 Summary

Each of the approaches presented here has advantages and inconvenients. Using a network simulator gives access to a large library of open source communication protocol models, but lacks accuracy. Hardware emulation is the most accurate technique, but is slow, and lacks scalability and logging features. Platform virtualization offers intermediate accuracy and seems promising, but is currently limited to TinyOS based systems and lacks some important features such as a power consumption model.

Ideally, the accuracy of emulation systems for firmware binaries would be combined with the execution speed of platform virtualization, and the configuration management, tracing capabilities and model libraries of modern network simulators.

3. INTERFACING EMBEDDED CODE WITH OMNeT++

This section describes a platform virualization method to integrate an embedded C code base for wireless sensor networks (named wisestack, for Wireless Sensor Stack) in the C++ OMNeT++ 4.1 simulator [2] and the MiXiM 2.0 modeling framework for wireless network simulations [1]. The following points are considered: general software architecture, embedded platform virtualization, and state consistency between the embedded and simulation codes. They are discussed in the following subsections.

3.1 Architecture

The following guidelines were adopted to enable the execution of embedded C code inside the C++ simulator:

- map each C embedded protocol module to a dedicated OMNeT++ encapsulation module;
- map method calls between C protocols modules to the OMNeT++ standard interface between modules;
- map the data structures used by the C code to OMNeT++ cMessage objects;
- for each host, save and restore the global variables used by the embedded C code in a Wisestack proxy module;
- introduce utility functions in the proxy module for units conversions (RSSI, time, network addresses);
- redefinition of key API functions of the C code and redefinition of various hardware related define statements to make the C code configurable at run-time;
- introduction of a set of global variables dedicated to information exchange between the embedded and simulation codes.

The extern “C” statement enabled the inclusion of C code in C++ code, and calling C functions from the C++ code.

3.2 Embedded Platform Virtualization

The embedded code relies on several assumptions concerning the system peripherals (in particular, the quartz crystal time reference, the flash memory and the radio transceiver) and on memory access (global addresses are accessed only by the current host), that do not hold when running within the simulator. This

section presents mechanisms developed to reproduce as closely as possible the environment in which the embedded code is executed.

First, many functions declared in header files have been redefined. Generally, the functions which required this are part of the Hardware Abstraction Layer. For instance, the uint32 tim_GetTime() function has been redefined. It returns the simulation time in seconds, multiplied by 1.024, and cast as an uint32. The conversion is needed because the considered embedded system uses a 32768 Hz quartz crystal as time reference and approximates milliseconds.

The embedded code depends on numerical constants set with define statements. To exploit the full possibilities of OMNeT++, and facilitate porting the firmware to other hardware platforms, these constants are not included when compiling for simulation, and the symbols are declared variables of adequate types instead. Pass-by-reference style indirection macros (#define GET_NAME(v)) are redefined as macros calling auxiliary functions by reference. The function then sets the adequate value.

A special difficulty was encountered with the MiXiM radio model, which only provides four states: RX, TX, SLEEP and SWITCH. The embedded code assumes a fifth state: ON, between SLEEP and RX or TX. Fortunately, MiXiM's design allowed the introduction of a RadioDetailed model subclassing the base radio.

The embedded system uses a system initialization function to initialize all global variables and C modules. It is not possible to call this function from OMNeT++, as this would cause for instance perfect synchronization between MAC protocol instances. Therefore, C modules starting embedded timers are initialized after a random initialization time (e.g. the MAC module).

3.3 State Consistency

Combining embedded and simulation code poses a specific challenge in data consistency. As OMNeT++ modules communicate with each other by exchanging cMessage objects, a module encapsulating an embedded protocol must convert the received information to the form expected by the embedded code.This consists mainly of copying all required information in the relevant C data structure. Inspection of state variables after executing embedded code allows the encapsulating module to decide whether to store, delete or send to other modules, the current cMessage. When sending cMessages, the encapsulating module stores in the object, the corresponding buffer index so that the receiving module can continue working with it, if it is also an encapsulating module.

4. CONCLUSION

This document described a novel method to execute embedded code within a network simulator. This method leads to fast simulations (orders of magnitude faster than cycle accurate simulation), enables the user to extract all variables of interest (using an already existing powerful logging and analysis system), allows modeling networks of all sizes, and combining seamlessly simulation and embedded communication protocols for the first time. The tool is currently used to port existing protocols to a novel radio, and to develop a novel protocol for future use on a hardware platform in development. The approach presented here led to the following improvements:

- reduced development complexity, by focusing on a single software stack;
- simplified network debugging, as many bugs can now be identified in simulation;
- reduced time to port the software stack to new radio transceivers;
- introduction of automated regression testing in our software development process.

Future work will consider even higher accuracy (by modeling SPI transfers), profiling to identify eventual simulation bottlenecks, and developing more regression tests.

5. ACKNOWLEDGMENTS

The research leading to these results was (partially) funded by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 231855 (sFly project).

6. REFERENCES

[1] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P.T. Klein Haneveld, T.E.V. Parker, O.W. Visser, H.S. Lichte, and S. Valentin. 2008 Simulating Wireless and Mobile Networks in OMNeT++: The MiXiM Vision. In Simutools '08, 1–8, ICST, Brussels, Belgium.

[2] András Varga. 2001 The OMNeT++ Discrete Event Simulation System. In Proceedings of the European Simulation Multiconference (ESM'2001), 319–324.

[3] Antoine Fraboulet, Guillaume Chelius, and Eric Fleury. 2007 Worldsens: development and prototyping tools for application specific wireless sensors networks. In Proceedings of IPSN '07, 176–185.

[4] J. B. Lim, B. Jang, S. Yoon, M. L. Sichitiu, and A. G. Dean. 2010 RaPTEX: Rapid Prototyping Tool for Embedded Communication Systems. ACM Transactions on Sensor Networks, 7(1).

[5] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. 2003 TOSSIM:accurate and scalable simulation of entire TinyOS applications. In Proceedings of the 1st international conference on Embedded networked sensor systems, SenSys '03, 126–137, New York, NY, USA.

[6] Mohammad Mostafizur Rahman Mozumdar, Luciano Lavagno, and Laura Vanzago. 2009 A Comparison of Software Platforms for Wireless Sensor Networks: MANTIS, TinyOS, and ZigBee. ACM Transactions on Embedded Computing Systems, 8(2).