Demo Abstract: Cross-level Simulation in COOJA

Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, Thiemo Voigt
Swedish Institute of Computer Science
Email: {fros,adam,joakime,nfi,thiemo}@sics.se

***Abstract*— Traditional WSN simulators are limited to simulating nodes at one single abstraction level. This makes system development and evolution difficult since developers cannot use the same simulator for both high-level algorithm development and low-level development such as device-driver implementations.**

The Contiki simulator COOJA allows for *cross-level simulation*, a novel type of wireless sensor network simulation that enables holistic simultaneous simulation at different levels. In COOJA one simulation can contain nodes from several different abstraction levels. These are the network level, the operating system level, and the machine code level.

We demonstrate a few different cross-level simulation scenarios using the COOJA simulator.

I. COOJA

Code development for wireless sensor networks is difficult and tedious [1], [2]. Network simulators can be used to simplify these development phases. Traditional sensor network simulators perform simulation at one fixed abstraction level such as the application, operating system or hardware level. The level at which the simulation is performed affects both the level at which software development can occur and the execution efficiency of the simulator. A simulator that simulates a particular sensor node platform at the hardware level enables the development of low-level software such as device drivers but at the price of longer simulation times and higher code complexity since low-level programming languages must be used. Conversely, a high-level simulator that does not model node hardware may provide short simulation times but only allows for development of high-level algorithms.

Since the need of abstraction in a heterogeneous simulated network may differ between the different simulated nodes, there are advantages in combining several different abstraction level in one simulation. For example, in a large simulated network a few nodes can be simulated at the hardware level while the rest are simulated at the application level. Using this approach combines the advantages of the different levels. The simulation is faster compared to when emulating all nodes, but at the same time enables a user to receive fine-grained execution details from the few emulated nodes.

Examples of traditional simulators include NS-2 at the network level, TOSSIM [3] for TinyOS at the operating system level and Avrora [4] at the machine code instruction level as shown in Figure 1. In contrast, our novel simulator for the Contiki operating system [5] COOJA enables *cross-level simulation* [6]: simultaneous simulation at many levels of the system. COOJA combines low-level simulation of sensor node hardware and simulation of high-level behavior in a single simulation. Furthermore, COOJA is flexible and extensible in

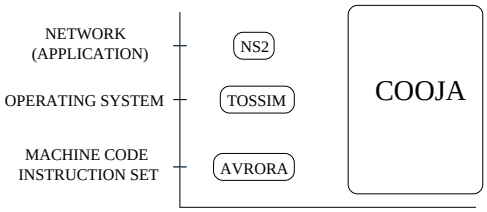

Fig. 1. COOJA can simultaneously simulate at several levels.

that all levels of the system can be changed or replaced: sensor node platforms, operating system software, radio transceivers, and radio propagation models. The simulator is implemented in Java, making the simulator easy to extend, but allows sensor node software, such as Contiki processes, to be written in C.

COOJA executes Contiki programs in two different ways. Either by running the program code as compiled native code directly on the host CPU, or by running compiled program code in an instruction-level TI MSP430 emulator. COOJA is also able to simulate non-Contiki nodes, such as nodes implemented in Java or even nodes running another operating system. All different approaches have advantages as well as disadvantages. Java-based nodes enable much faster simulations but do not run deployable code. Hence, they are useful for the development of e.g. distributed algorithms. Emulating nodes provides more fine-grained execution details compared to Java-based nodes or nodes running native code. Finally, native code simulations are more efficient than node emulations and still simulate deployable code.

Each simulated node in COOJA has three basic properties: its data memory, the node type, and its hardware peripherals. The node type can be shared by several nodes, and determines properties common between these nodes such as the simulated software. The hardware peripherals of simulated nodes are in COOJA called *interfaces*. Interfaces enable the Java simulator to simulate external events such as incoming radio traffic or external interrupts and to trigger events such as LED changes. Interfaces also represent properties of nodes such as a simulated node position. The node data memory represents the current node state, and changes over time as the simulated software is allowed to execute.

When a simulation is running, nodes are sequentially allowed to act for a short time each. The node interfaces are polled both before and after the node software is allowed to execute. When all nodes have acted the simulated time is increased a fixed amount of time.

All interactions with simulations and simulated nodes are

performed via *plugins*. An example of a plugin is a simulation control that enables a user to start or pause a simulation. Both interfaces and plugins can easily be added to the Java simulator, enabling users to quickly add custom functionality for specific simulations.

Each simulation in COOJA uses a radio model that characterizes radio wave propagation. The radio model is chosen when a simulation is created. This enables a user to, for example, develop a network protocol using a simple radio model, and then test it using a more realistic model, or even a custom made model to test the protocol in very specific network conditions. New radio models can easily be added to the simulation environment. COOJA supports, except from a completely silent model, two different radio models. The first is a unit disk graph radio model that has configurable interference and transmission range parameters. The second model is based on ray-tracing and supports multi-path effects with radio absorbing materials.

II. Cross-level simulation

COOJA supports three different abstraction levels. Nodes simulated at the **appliction or networking level** are implemented in Java. Without any connection to Contiki, nodes at this level can be useful when prototyping high-level algorithms which when tested and evaluated can be ported to deployable sensor node code. Other examples of other useful Java nodes are radio packet loggers, and traffic generators which are discussed in further detail in Section III.

Nodes simulated at the **operating system level** execute deployable Contiki code, but compiled as native code. An entire Contiki system, including the core, pre-selected user processes, and a set of special simulation glue drivers, is compiled to a shared library. The library is loaded and controlled from the Java simulator via Java Native Interfaces (JNI). COOJA exploits the event-driven approach used in Contiki by calling the loaded system in a way so that each simulated node handles one event each every simulation loop.

The library memory defines the state of a node. It is copied to, and later is fetched back from the library to the simulator when the node handles an event. The interfaces of the simulated nodes communicate with the Contiki system by manipulating this data memory. COOJA supports loading several different libraries, these are loaded from different node type objects. And since the library code memory is read-only, all nodes of the same type share the same simulated code. The data memory of the library may however differ between each simulated node.

The machine-code level abstraction level in COOJA is enabled by connecting a Java-based microcontroller emulator to the simulator. It emulates an ESB node [7] (MSP430) at the hardware level. The emulated nodes are controlled in a similar way as the native code nodes. Each simulated node is allowed to execute for maximum a fixed period of time or long enough to handle one event. Events are then, by using the current node memory, transferred via the hardware interfaces to and from the simulator.

III. Evaluation

We have previously demonstrated that, using cross-level simulation in COOJA, it is possible to maintain advantages specific to the different levels [6]. These advantages include processing times and memory requirements of higher abstraction levels and enabling access to fine-grained details of lower levels. For example it is possible to simulate 2000 nodes in a cross-level simulation (20 emulated, 1980 pure Java) in the execution same time as a simulation with only 50 emulated nodes.

Another useful example of cross-level simulation is the simplicity of adding nodes with specific tasks. We implemented a radio traffic generator node at the application level, i.e. in Java. As it is much easier to implement Java code than low-level C code, the implementation only took a few hours. The traffic generator periodically transmits radio packets at a high rate, thus efficiently disturbing surrounding radio traffic.

The main advantage of adding a new traffic generator node compared to “manually” disturbing selected radios from a plugin or a radio medium, is the flexibility it offers. Traffic generator nodes can be added and removed at run-time just as easily as any other simulated node. Further, the user can interactively change the radio channels the traffic generator nodes disturb.

Acknowledgment

This work was partly financed by VINNOVA, the Swedish Agency for Innovation Systems, and the European Commission under contract IST-004536-RUNES.

References

[1] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, “Sympathy for the sensor network debugger.” in *ACM SenSys*, 2005, pp. 255–267.

[2] G. Wittenburg and J. Schiller, “Running realworld software on simulated wireless sensor nodes,” in *Proc. of the ACM Workshop on Real-World Wireless Sensor Networks (ACM REALWSN’06)*, Uppsala, Sweden, June 2006.

[3] P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: accurate and scalable simulation of entire tinyos applications,” in *Proceedings of the first international conference on Embedded networked sensor systems*, 2003, pp. 126–137.

[4] B. Titzer, D. K. Lee, and J. Palsberg, “Avrora: scalable sensor network simulation with precise timing,” in *International Conference on Information Processing in Sensor Networks (IPSN)*, 2005.

[5] A. Dunkels, B. Grönvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, Nov. 2004.

[6] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, “Cross-level sensor network simulation with cooja,” in *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Tampa, Florida, USA, Nov. 2006. [Online]. Available: http://www.sics.se/nes/osterlind06crosslevel.pdf

[7] J. Schiller, H. Ritter, A. Liers, and T. Voigt, “Scatterweb - Low Power Nodes and Energy Aware Routing,” in *Hawaii International Conference on System Sciences*, Hawaii, USA, Jan. 2005.