DEMO: RealMachine
Graphical programming for WSNs

K. Terfloth, H., Ritter, J. Schiller
Freie Universität Berlin
Takustr. 9
14195 Berlin
+49 (0) 30 838 75123
terfloth|hritter|schiller@inf.fu-berlin.de

ABSTRACT

Programming applications for wireless sensor networks usually involves expert knowledge of hardware, distributed networking and the associated problem domain. Realizing even a simple task like periodic gathering of environmental data results in writing embedded C code. After compilation, the new binary image has to be deployed on every node in the network, which cannot, to the best of our knowledge, be done on a multi-hop path by now. In order to avoid sending huge images over the small-scale networks, a virtual machine that is able to interpret byte-code is a feasible approach. In this demo we will show the usage of RealMachine, a new kind of virtual machine developed for the ScatterWeb platform [1]. RealMachine has been designed to enable graphical programming in a safe and user-friendly environment, with intuitive tools and a compact byte-code image. Furthermore, we will demonstrate a very convenient way to reprogram a deployed network by infecting a single node. This node initiates the dissemination of the new byte-code and infects the complete network on a multi-hop path.

FEATURES

The IDE of RealMachine provides means to set up new projects, to compile them directly into the specific byte-code and to load them onto an attached node. Figure 1 gives an example. Applications can easily be composed from modules similar to flowchart languages allowing users already familiar with symbols typical for this syntax a fast adaptation. Icons can represent parts of the sensor nodes, for example a temperature sensor or current battery level, as well as user components implemented by a user. To build a new application the user picks an object with the mouse, drags it into the workspace of RealMachine and drops it there. A new instance of the kind of object is created and ready to either get connected to another component, or to be plugged into a method that shall be invoked. Since objects may have input and output variables they can be wired respectively on demand.

After a user has created a task by clicking it together, he can choose to make a new byte-code image and replace an old one on a sensor node. Therefore, the user may make use of the ScatterFlasher, a USB device that sends the byte-code over the air to the nodes. The new application can be routed on a multi-hop path to every single node of the deployed network without a single touch and without any cables.

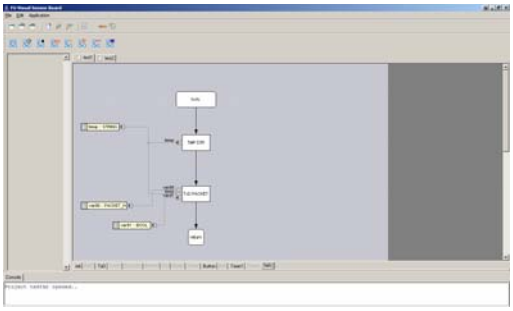

Figure 1: Graphical programming of the RealMachine

REAL MACHINE APPROACH

While the presented solution is oriented at the ideas of a virtual machine we call it RealMachine as it differs from a classical virtual machine approach. The RealMachine does not follow the vision of presenting to the application programmer a full-featured, though virtual, processor. The main design goal was to build a virtual machine allowing in principle that a RealMachine-application runs as fast as a compiled application written in C and compiled for the same target hardware.

This goal is achieved by exposing complex functions of the firmware to the byte code level instead of re-implementing them by basic commands of a virtual processor. For example, a building block of the IDE that can be used to send a packet over the network is directly mapped to the function pointer of the respective firmware function that is used for sending a packet. Even an application that is written in C and compiled for the target would not perform faster as it used the same function.

The RealMachine and its IDE are especially adapted to the typical programming paradigm of embedded systems like wireless sensor nodes. The RealMachine provides entry points for methods that are called if an event of a pre-defined type (like a special sensor event, or incoming data packets) occurs. The IDE allows the user to graphically program the method for each possible sensor event, while the RealMachine handles the assignment of these methods to the respective firmware callback functions. Timer events are handled in a similar way.

The demonstration covers:

- Graphical programming with the IDE
- Compiling, downloading, flashing at one click
- Over-the-air reprogramming over multiple hops

The demonstrator consists of 20 sensor nodes, a laptop and an USB gateway to the sensor network. It allows hands-on programming for conference participants.

[1] ScatterWeb project, http://scatterweb.mi.fu-berlin.de