A VIRTUALLAB APPROACH FOR WIRELESS SENSOR MOTES AND WIRELESS SENSOR NETWORKS

Anghel V. CONȚIU, Adina ȚOPA, Vasile T. DĂDÂRLAT
Technical University of Cluj-Napoca, Str. Ctin Daicoviciu nr. 15, 400027 Cluj-Napoca, Romania
E-mail: Vasile.Teodor.Dadarlat@cs.utcluj.ro

Abstract: The VirtualLab project was developed to fill the need of exploring the world of sensors, to make it available to younger students and to increase their interest in studying real sciences like physics, electronics or computer science. It is also a solution for low budget laboratories, making the motes available remotely.

Key words: *Mote, Radio, Remote, Sensor, Wireless.*

I. INTRODUCTION

Wireless sensor networks provide features for monitoring different aspects of the environment starting with the humidity and ending with moving elements.

The research labs are in their early phase of developing the devices, but they have already made enough progress in order to present them in schools and to capture the student's attention. A lot of challenges are yet to be considered [1] for Wireless sensor networks regarding the real-world protocols, real-time data, power management, programming abstractions, security and privacy still require a lot of attention from the developers.

The VirtualLab project was developed to fill the need of exploring the world of intelligent sensors, considering the fact that nowadays the laboratories must invest a lot of money to keep up with the latest technologies and to make them available for their students. The VirtualLab represents a very handy solution for low budget school labs or low budget research labs, until they are able to buy the devices.

This paper is devoted to the presentation of the virtual laboratory concept that was developed by us in the VirtualLab project. We used the nesC programming language (a component oriented extension of C), the TMote Sky sensor motes developed by Moteiv and a Java API that was specially designed for the applications using sensor motes.

Section II describes the developed design concept, the mote components that were created to be wired to user's own components (II.1), the server application together with its specific features (II.2) and the client application that was developed for a high level interaction between the users and the remote motes (II.3). The experimental results are further on presented on section III. Section IV is devoted to the conclusions of our work and considerations for the future.

II. VIRTUALLAB DESIGN CONCEPT

The application is made out of three major components that have the purpose of getting the user in a virtual contact with the motes. This implies remote connectivity over the local area network or over the internet. The entities are:

- the motes inside the laboratory;
- the server application running on the computer that the motes are connected to;
- the client application - it represents the user's tool for handling the motes.

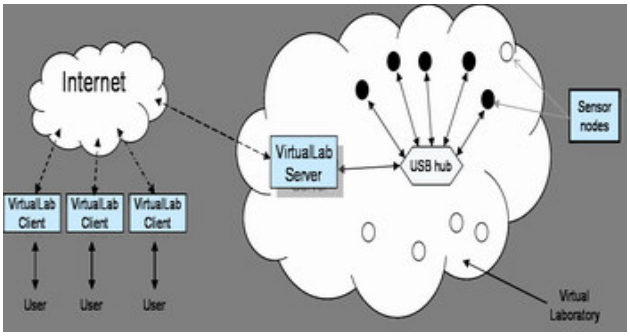

Figure 1 – The architecture of the VirtualLab project

Manuscript received February 19, 2008; revised April 17, 2008

The entities must benefit of a good communication in order to provide the user with reliable data on one hand and for the motes to understand real time settings that the user sends via the server. The application needs speed in data transmission, but very important, it needs reliable data to be received by one entity or another. This is why the server and the client applications were developed to communicate using the socket technology [2] and the TCP/IP protocol. The mote – PC communication is made by USB, via the PC's serial port.

II.1. THE MOTES

The application uses the Tmote Sky motes developed by Moteiv [3]. A sensor node is made up of four basic components: a sensing unit, a processing unit, a transceiver unit and a power unit [3]. They use the TinyOS operating system [4] and [5], an operating system that is able to deal with the several constraints implied by the devices. The applications that run on the motes were developed using the NesC programming language, a component oriented extension of C [6].

There are two important types of applications that are installed on the motes, in order to provide the required features:

- The first one is the Deluge application; it is able to handle multiple applications that are installed on the mote [7] on one hand and it is able to program a network of motes on the other hand, using its radio message epidemic [8].
- The second type of applications is represented by the ones that were developed as part of VirtualLab (LedsObserver,USB_Autodetect,OscilloscopeTmote Sky) together with the applications that are created by the user and downloaded on the motes. The user is provided with ease in monitoring his applications by wiring them to our components. This way he will benefit of all the features that our applications provide. The user's applications are handled by Deluge.

The **Deluge** application can be considered as an intermediary element, being placed between the operating system and the user's application as seen in the next figure:

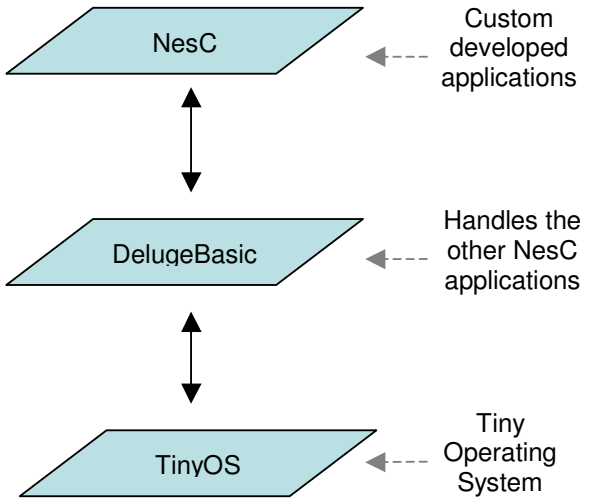

Figure 2 – The stack of application types

. The motes are able to receive two types of messages, considering the two types of applications, while they are approached by the VirtualLab. These are:

- Deluge level messages – commands that are directly addressed to Deluge and they handle the applications that are currently installed on the mote or inject new applications. Examples of this kind of commands are: "application list", "erase", "reboot & run", "inject".
- Application level messages – commands that are addressed directly to the currently running application on the mote. These commands are created by the user with respect to the application that is running, taking into consideration the type of messages possible to be received.

VirtualLab provides some nesC components in order to help the user retrieve important pieces of information about the mote. One of those components is the **LedsObserver.** It observes the state of the leds at some custom defined time intervals.

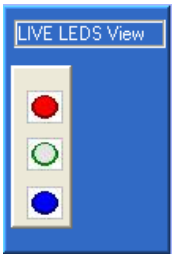

Figure 3 – The "Live Leds View" The red, green and blue leds inside the window are activated at the same time

as the corresponding leds on the mote

This component is important because by sending the data to the server the leds state will reach the VirtualLab client application, making its graphical user interface able to actually display the on/off state of each led in real time. It is well known the fact that once a nesC application is installed on the mote and it is not supposed to send any data to the PC, the only way to debug it is to observe whether the mote's leds behave as they should. The big advantage of the LedsObserver component is that once it is wired to the user's application, it allows the user of the VirtualLab to debug its new nesC application by observing on its monitor the way the mote's leds change their state in real time, so he can observe the way its application is working and take the necessary actions accordingly.

The user can start or stop the mote from sending its leds state, meaning that its timer will start/stop until the user will send another command to the component.

The **OscilloscopeTmoteSky** application deals with all the sensors on the Tmote Sky modules and communicates the sensor values over the radio to a base station running the TOSBase application and by UART to the PC the mote is connected to (ie. the PC running the VirtualLab server application).

Here are the currently supported sensors: Sensirion Relative Humidity Sensor, Sensirion Temperature Sensor, Hamamatsu Photo Synthetically Active Radiation Light Sensor, Hamamatsu Total Solar Radiation Light Sensor, TI MSP430 Internal Temperature Sensor, TI MSP430 Internal Voltage Sensor [3]. Each sensor is assigned a channel for sending its data.

In order to send the sensor readings to the PC or to the other motes by radio, the OscilloscopeTmoteSky component is wired to the CommOscope component. The same component is used to change the application's parameters. While the application is working, the user can change the sensor that is currently reading the data, the number of sample readings the application will send and he can also adjust the time interval the readings are made. The user can use its VirtualLab client to set those parameters and to send the command to the server. The server creates a **VLabOscope message type** containing the new parameters and sends it to the mote. The mote receives that message by UART using the CommOscope component and calls the setParameter (sensor, readings, freq) command through the ComponentComm interface. The OscilloscopeTmoteSky component provides this interface and implements the setParameter (sensor, readings, freq) command. The mote accepts the sensor parameter setting as a coded number, so it is decoded by the NesC application in order to establish the sensors that must be read.

The OscilloscopeTmoteSky component is also wired to the LedsObserver component so the user can actually see the state of the leds while the OscilloscopeTmoteSky application is running.

The **USB_Autodetect** is a component that is able to check whether the mote is connected to the PC by USB or whether it works as a stand alone mote, powered by batteries. The component is based on the interruptions that result when the user connects/disconnects the mote from the USB. Wiring this component to the application provides the user with important information related to this aspect. This way he can develop his entire NesC application telling the mote to perform as a stand alone mote (start communicating by radio because there is no USB connection) or to perform as a mote that is connected to the PC and send data by UART or by both radio and UART, according to the user's needs. More work can be done for the interoperability between the USB_Autodetect component and Deluge's epidemic feature

.

II.2 .VIRTUALLAB SERVER

The server uses the TinyOS Java tool chain in order to provide PC – mote communication. Two of the most important objects coming from the tool chain are the PhoenixSource (built on BuildSource) and the MoteIF. They provide the basics for this type of communication. A PhoenixSource builds upon a PacketSource to provide the following features:

- Automatic reading and dispatching of packets: registerPacketListener and deregisterPacketListener;
- Automatic source restarting (via setResurrection); it is off by default.

PhoenixSources are threads and hence they need to be started. The default MoteIF constructor uses the MOTECOM environment variable to determine how Java application connects to the mote; VirtualLab Server application hides this setting from the user and sets the variable automatically.

The **FreshMessageParser** object determines the type of objects coming from the client; it acts as a handler by creating tasks and assigns them to other objects.

The object representing the message type the MoteIF sends and receives from the mote must be fully compatible with the message type the mote expects; these objects are built with respect to the .h files from the NesC code. In order to archive this, one can use the **MIG** tool [9]. It will automatically generate a Java class based on the .h file. MIG reads the NesC *struct* definitions for message types in the mote application and generates a Java class for each

message type that takes care of the gritty details of packing and unpacking fields in the message's byte format. Using MIG saves the developer from the trouble of parsing message formats in its Java application.

The motes are assigned a COM port on the PC when they connect to it and the MOTECOM environment variable must be set in order to communicate to the mote. Depending on the way VirtualLab's MoteIF messages are sent / received from the mote and the way the TinyOS Java tool chain (with its own MoteIF objects) approaches the COM port in order to send / receive messages, several problems needed to be solved. As the mote is communicating with the computer using a certain MoteIF object, the port becomes busy and another application, with its own MoteIF object can not begin communicating on the same port. VirtualLab uses three MoteIF objects in order to provide all its features. There are the VirtualLab's MoteIF, the Deluge's MoteIF object (for deluge level messages) and another MoteIF that is used by the "Listen" feature. Part of the problem is approached by the FreshMessageParser object, because it is the handler for the client's request so it is assigned the responsibility of making sure that the request can be fulfilled. There are three main problems that had to be solved:

- providing transparency for setting the MOTECOM environment variable, meaning that it must be set automatically;
- creating the conditions for sending MoteIF objects as application level messages to the mote;
- sending deluge level messages or Listen commands handling their own MoteIF object.

A new process is built for a command. The FreshMessageParser object passes the job to an object that handles the command. The first step for handling the problem is to assign a different thread for the deluge level commands; each thread will create a new process that will have set the necessary environment variables. In order to archive this, the ProcessBuilder and Process [10] classes are used. Before starting the thread, all the necessary parameters for the new process are set using the ProcessBuilder and the first step inside the thread is to start the process. This way, before starting the process, each message that the client sends to the mote will have all the necessary conditions fulfilled on the server side, providing its own "environment" (the new process) where the MOTECOM variable is correctly set, so that everything will go as planned. A watch timer is started above each of this type of processes and it will close the process if something goes wrong.

II.3. THE VIRTUALLAB CLIENT

The goal of the VirtualLab application is to make the users able to use the sensor motes in such a way that even though they don't actually have their own devices, they can remotely explore the new technology, develop their own application, download it on the motes, test the mote's limits and increase their interest in this area.

The client application provides the right pieces of information that will get to the mote in the end and it must also be able to receive data, interpret it and display it in an efficient and understandable way for the user.

Serializable objects are used in order to communicate to the server. Once the user has selected the settings he wants to send to the mote by using the GUI, the client application creates the necessary data structures that will be sent to the server, containing the user's settings. The data structures are known to the server, so it will know how to handle the incoming objects in order to transform them into commands that will be sent to the mote.

III. EXPERIMENTAL RESULTS

In order to test the applications, the virtual laboratory was provided with a computer and motes. The computer runs the VirtualLab server application and it is connected to the internet so the communication with the client can be achieved. The motes run the nesC applications and they receive commands from the client through the internet and through the server application.

The following hardware was used for testing the applications for the:

- **VirtualLab Server:** Desktop PC:Pentium III, 1.8 Ghz, 256 RAM, Laptop PC Athlon XP 2500, 1800 Mhz, 512 RAM;
- **VirtualLab Client:** Laptop PC Athlon XP 2500, 1800 Mhz, 512 RAM;
- **NesC applications**: Tmote Sky wireless sensor motes

The OscilloscopeTmoteSky application ran on the motes being wired to LedsObserver, Usb_Autodetect.

The Usb_Autodetect component of each mote acknowledged the fact that its mote was connected through USB and the communication was ready to be started through USB. After that, the user has the possibility to use its client application in order to check the list of connected motes, to select a mote and to start sending commands to it.

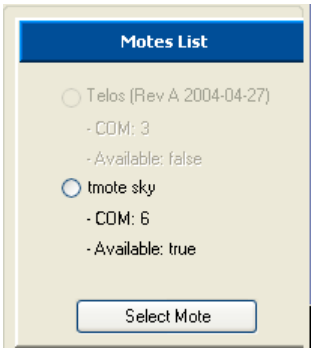

Figure 4 - the "Mote List" window was developed so the user can select the mote he wants to address inside the virtual lab.

After selecting a mote, the user can start sending deluge level messages to the mote, and see the list of installed NesC applications, remove or inject applications. He can also reboot the mote.

Having the OscilloscopeTmoteSky application installed, the user selects the sensors he wants to read using the image panel by clicking on the right sensor, as seen in the next figure:

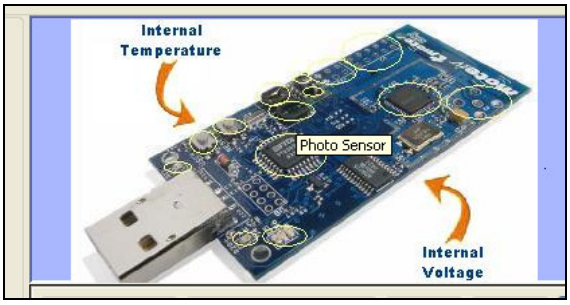

Figure 5 – the image panel; the sensors are marked by the yellow circle and a tool tip will be displayed when the mouse pointer is placed over one of the sensors. The sensors that are on the other side of the mote are marked by the orange arrows. A sensor is selected when the user clicks on it.

The client application asked for the sensor readings to be performed in the following manner:

- the sensor readings must be executed using samples of 4 readings;
- the samples must be taken at intervals of 726ms.

The experiments provided the following results for each of the available sensors:

The **TmoteSkyOscilloscope** application:

- Internal Voltage readings: a sample of 4 readings, taken at 726ms interval: 2.17mV, 2.17mV, 2.17mV, 2.17mV;
- Internal Temperature readings: a sample of 4 readings, taken at 726ms interval: 26.56°C, 26.56°C, 26.56°C, 26.56°C;
- Outside Temperature readings: a sample of 4 readings, taken at 726ms interval: 28.78°C, 28.85°C, 29.05°C, 28.91°C;

The **Listen** command:

One mote runs a NesC application that counts continuously and sends each value to our mote that runs the TOS_Base application. The TOS_Base application receives messages by radio and forwards them by the serial port to the computer running the VirtualLab server application.Here are the messages that the client application receives in the end:

Listen: 04 01 08 9F FF FF FF FF 04 7D 9F 01 01 00
Listen: 04 01 08 A0 FF FF FF FF 04 7D A0 01 01 00
Listen: 04 01 08 A1 FF FF FF FF 04 7D A1 01 01 00
Listen: 04 01 08 A2 FF FF FF FF 04 7D A2 01 01 00

The values are sent by the mote in the little-endian coding and one can see here that the client receives the hexadecimal values from 019F to 01A5.

The LedsObserver feature was created to work close to perfect accuracy, although it has a disadvantage, as its performance depends on the networks delays because of the remote connection between the computers running the VirtualLab Server application and VirtualLabClient application. It might be influenced by a low performance computer because of the GUI's requirements.

IV. CONCLUSIONS

Openness is the propriety of a system to be easily extended and modified. More work has to be done in including MIG as part of VirtualLab in order for it to provide the new Java classes based on the structures in the ".h" file that is used by the NesC application.

A stronger support for radio communication can be provided by creating a component that will work together with our current USB_Autodetect component, and by researching the customization of Deluge message epidemic.

The VirtualLab concept was implemented having three important entities (the server, the client and the nesC applications) that are not coupled, having their own, well defined responsibility, and providing the possibility to be extended. The NesC components that were provided offer students the possibility to become familiar with an area of computer science that seems difficult and not very attractive at first sight. They have the chance to enjoy experimenting low level languages and programming hardware

components. VirtualLab can also be used together with other types of motes like mica, mica2 or telos, as long as they support the TinyOS operating system. Some of the new motes explore some Java platforms, yet the principles are the same, so VirtualLab together with NesC and TinyOS can still be used for education purposes.

To develop the project's features, several other elements were implemented: it uses the TCP/IP connection for supporting the new concepts of Deluge level messages and application level messages; messages that are also sent between the client and the server are also more reliable. The server was implemented to be multithreading, and to provide message handling and serial connection with the motes; some important features that are specific to VirtualLab's server are the port handling and message object handling in order to avoid communication problems and the watch timers for avoiding process stagnation; it uses MIG (message interface generator) for server – mote communication objects and TinyOS Java tool chain for a proper server – mote communication.

One of the targets of VirtualLab was to provide a high level interaction between the user and the motes, so a very friendly graphical user interface was provided on the client side; the necessary tools were implemented for displaying the list of available motes inside the VirtualLab, allowing the management of the installed applications and providing flexibility; more than that it provides the necessary nesC components and graphical user interface elements for the user to be able to do real time debugging on its own fresh NesC applications that he installed on the motes using the VirtualLab, while a lot of nasty operations are transparently executed in the background.

REFERENCES

[1] Stankovic, J.A., *Research Challenges for Wireless Sensor Networks*, ACM Press, July 2004
[2] Ken Backlawski, *Java Socket Tutorial, (*available at http://www.ccs.neu.edu/home/kenb/com1335/socket_tut.html)
[3] Tmote-sky-datasheet (available at www.moteiv.com)
[4] Philip Levis, "TinyOS Programming", Feb 14th 2006
[5] Jonathan Hui, "TinyOS Network Programming", July 25th 2005
[6] David Gay, Matt Welsh, David Culler, "*The nesC Language: A Holistic Approach to Networked Embedded Systems"*, ACM Press, May 2003
[7] Deluge: TinyOS Network Programming (available at http://www.cs.berkeley.edu/~jwhui/research/deluge/)
[8] The Deluge manual
[9] The MIG tool (available at http://www.tinyos.net/tinyos-1.x/doc/nesc/ncg.html)
[10] Java API (available at http://java.sun.com/j2se/1.3/docs/api/java/lang/Process.html)