NTUPreter: High-Level Structured Programming Platform for Wireless Sensor Networks

Yu-Cheng Norm Lien and Wen-Jong Wu
Department of Engineering Science and Ocean Engineering, National Taiwan University, Taipei, Taiwan
Email: yclien@ntumems.net, wjwu@ntu.edu.tw

***Abstract*—Wireless sensor networks have gained world-wide attention in recent years, researches were proposed to address challenges of sensor network programming. Making programming simple and flexible encourages users without programming background to learn and adopt wireless sensor network. We present the design of the NTUpreter, a remote programmable platform for wireless sensor network. This platform allows users to program senor nodes through a high-level structured programming language interpreter and supports remote reprogramming therefore reduces the cost and effort of deployment. We also developed a language extension for the interpreter to access low-level hardware and wireless functions. An integrated development environment (IDE) with steep learning curve is also introduced in this work.**

***Index Terms*—structured programming, micro-interpreter, Wireless Sensor Network**

I. Introduction

Wireless Sensor Networks (WSNs) have been widely utilized in the field of sensing and measuring in recent years. They are consisted of resource-limited sensor nodes that are able to perform sensing, computing, actuating, and wireless communicating tasks. Their low-power, short-range and low-price characteristics made them particularly suitable for long-interval and large-scale deployments for environment monitoring. Applications of WSNs cover many disciplines including agriculture, military, home automation, industry and science research.

Two major issues are to be addressed in applying WSNs. One is that the technical barriers of entry for wireless sensor network programming are too high for application domain experts [1]. Wireless sensor network programming requires knowledge to program, compile and load executable files to the sensor nodes. It generally requires additional knowledge of specific embedded operating systems and hardware circuits. This makes the utilization of WSNs be restricted to experts with background knowledge. Another problem of WSNs is the reprogramming procedure of deployed sensor nodes. Typically, sensor nodes should be programmed (flashed) using wired (RS-232 or JTAG, etc.) communications etc. This makes reprogramming in massive deployed sensor networks difficult. Moreover, wireless sensor network development environments currently requires users to code, compile and upload executable files by separate editors or command lines. These complicate procedures confuse users. In most cases, learning the procedures is often not their purpose.

The objective of this work is to decrease the programming effort by addressing the two mentioned issues. Therefore, experts from different domains of science and technology may benefit from this platform. In our work, we developed an interpreter that runs on sensor nodes to support high-level structured programming. Second, we provide users an integrated development environment with a clear and clean user interface. Last, we utilize a seamless downloading technique to allow smart sensor nodes be reprogrammed over the air.

The rest of the paper is organized as follows: First, we give an overview of the related researches. Then we describe the components to give an overview of our system. We describe the improvements of our interpreter implementation in detail. We then outline the over-the-air reprogramming function of our platform. Finally, we describe the design of our platform IDE.

II. Related Work

Several works were done to simplify the procedure of wireless sensor network programming. Some of the works tempt to place a virtual machine (VM) into every wireless sensor node and program each one by using an assembly language [2]. Some are based on these kinds of virtual machines and further provide the ability to run as a subset of a Java Virtual Machine (JVM) [3]. These efforts allow programmers to be able to program sensor nodes with java language. However, the java syntax is complicated that makes java unsuitable for domain experts. On the other hand, some efforts are made to bring BASIC Language to microcontrollers programming [4]. BASIC Stamp is a microcontroller with BASIC interpreter built into its ROM. It supports common microcontroller functions such as serial communication and I2C communication. However, the programming involves too many low-level details and there is no support for IEEE 802.15.4 communication.

Efforts mentioned above try to make microcontroller programming easier, while some works intend to make existing wireless sensor network programming easier to manipulate by encapsulating the complex networking details by using application programming interfaces

Manuscript received November 2, 2013; revised January 5, 2014.

©2014 Engineering and Technology Publishing

doi: 10.12720/ijeee.2.2.138-142

(APIs) [5]. Since domain experts are not interested in how the data are sent (data aggregation, routing path, collision, etc.), exposing low-level detail information may lead to more confusion and hold back the learning intention.

uBASIC is a basic interpreter developed by Adam Dunkels that is released under a 3-clause-BSD-style license [6]. It is an unstructured basic interpreter designed for microcontrollers and offers a subset functionality of general BASIC interpreters. uBASIC has been ported and extended to wireless sensor nodes by Miller et al. to lower the programming barrier for domain experts. Experiments comparing BASIC language and an event-based scripting language TinyScript are done and they have shown that BASIC is more suitable for novices than TinyScript [1]. Although BASIC interpreter is brought to wireless sensor nodes, the unstructured attribute strongly hinders programmers from studying and utilizing it [7]. Moreover, the functionality of the proposed system is still distant from applying it to general sensor network applications.

III. System Overview

According to our objectives, we proposed a system including the followings:

- Smart sensor node hardware
- An embedded operating system
- An interpreter for high level programming
- An IDE for development and deployment

A. Smart Sensor Node Hardware

The hardware platform used in our work is the National Science Council & National Taiwan University Wireless Sensor Network Modules (NSC NTU WSN). These modules majorly contain two types of sensor nodes: SuperNode and SimpleNode. In this work, we only use SuperNode. It is approximately a clone of Tmote Sky [8]. However, TI CC2420 IEEE 802.15.4 wireless transceiver of Tmote Sky is replaced by the UBEC UZ2400 IEEE802.15.4 wireless transceiver. Also, SuperNode does not contain the DS2411 serial ID IC and the M25P80 Flash IC.

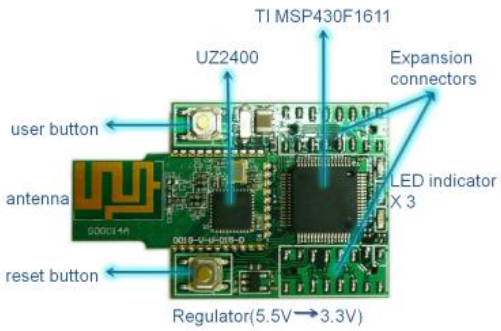


Figure 1. SuperNode sketch.

As mentioned before, smart device hardware is where the limit for the device is defined. In our case, we choose to use SuperNode from the NSC NTU WSN Modules. SuperNode uses Texas Instruments MSP430 ultra low power microcontroller, whose low power consumption characteristic and fair computing power make it extremely suitable for applications such as Wireless Sensor Networks.

B. Embedded Operating System

In wireless sensor networks, software on sensor nodes may or may not contain an operating system (OS). Sensor nodes with operating systems may have the ability to perform multi-threading, resource allocating, security enhancing, etc. Since modern sensors tend to have more resource including processing power, memory and flash storage. They are now more affordable to contain operating systems.

Well known operating systems for WSN includes TinyOS (University of California, Berkeley) [9], Contiki (Swedish Institute of Computer Science, SICS) [10], etc. Detail comparisons between wireless sensor network operating systems may be found in [11]. TinyOS is an event driven OS introduced by UC Berkeley. It uses nesC as the developing language which may advance the learning effort to build up applications in this environment. On the other hand, Contiki, uses C language as its developing language. In this work, we use a modified version of Contiki that is ported to our NSC NTU WSN platform.

For our purpose, the work is focusing on improving neither the performance of the operating system nor the networking ability. Instead, we utilize existing operating systems to offer stable and robust underlying services and concentrate on building our service beyond this environment. By taking benefits from open source projects, we focus on the designs of the interpreter.

C. Interpreter

An interpreter is a program that is able to execute scripts that is written in a specific programming language. An interpreter usually executes the written script directly. Although, in some cases, interpreters translate scripts into intermediate codes and execute them.

In either ways, interpreters should do the translation every time scripts are executed. This leads to consuming more computing resources and therefore more power consumption in finishing the same task, compared with what compilers consume. But in contrast, interpreters also have their benefits. Interpreters are interpreted directly from scripts therefore programs are mostly more portable than those generated by compilers. In our work, for fast prototyping and analysis, we choose portability and development cycle rather than performance.

BASIC (an acronym for Beginner's All-purpose Symbolic Instruction Code) language, as the name given, is a high-level programming language particular suitable for beginners. Corresponding to our objectives, this programming language should allow domain experts omit the complexity of smart sensor node programming and further provide an environment that offers dependable fundamental wireless communication and computation capability. Since BASIC language is well-known for its straightforwardness either in learning or practicing, it is selected to be the targeted language of this work. To

©2014 Engineering and Technology Publishing

make this interpreter actually work, it is essential to build a BASIC interpreter on a wireless sensor network operating system. This interpreter will operate as an overlying layer above the OS. Architecture of the interpreter node is shown in the following figure (Fig. 2):

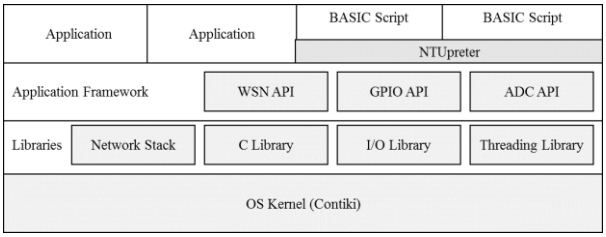


Figure 2. SuperNode sketch.

D. IDE for Development and Deployment

There are many factors that may affect users whether to use a development kit or not. One of the critical factors for software developers is the Integrated Development Environment (IDE). An integrated development environment is an application that provides friendly and versatile environment to computer programmers for software development. Because of its convenient features that eliminate the work of mode switches, it may help developers achieve higher productivity by reducing effort.

IV. Interpreter Design

Interpreters should be capable to wrap hardware dependent functions into native code routines. Therefore, native code routines may be called by lines of scripts. For example, wireless transceiver functions and flash storage functions may be called by one or two simple line of scripts. According to our objectives, the scripting should be as easy as possible to allow novice users to program with less effort. The structure of our interpreter is shown in Fig. 3:

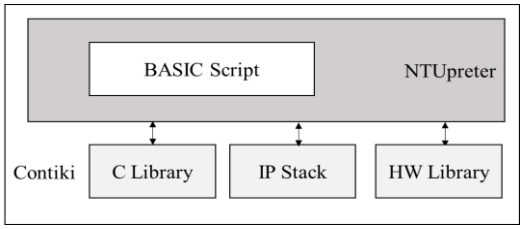


Figure 3. Structure of our interpreter.

The interpreter in this work is based on the uBASIC interpreter which is originally written by Adam Dunkels [6]. This interpreter is a BASIC interpreter that provides unstructured BASIC programming. Based on uBASIC interpreter, we transplanted uBASIC to our Contiki-ready SuperNode to obtain a BASIC interpreter for smart devices-NTUpreter. This is done majorly by making NTUpreter an application of the embedded operating system. Every single application runs as a protothread in the Contiki OS [10]. In our case, we include part of the wireless function as well as hardware function source code into our interpreter. These source codes are further wrapped into functions that may be called directly by our interpreter. To allow the interpreter to call these functions, new syntaxes for triggering these routines are necessary. As a result, we designed a wireless extension and a hardware extension for our BASIC interpreter. These extensions provide new syntaxes and new functions for our interpreter to suite our usage and purpose. The following shows the major improvements in our proposed ntuBASIC interpreter.

Structured BASIC interpreter

The main difference between structured BASIC and unstructured BASIC programming is that structured BASIC provides procedure-oriented programming. The line numbers are generally replaced with labels and procedures to advance the flexibility and the readability of scripts.

Supporting Input Syntax

uBASIC is a well-designed program that offers most of the functions that a BASIC interpreter should have. However, an important syntax input is missing. In wireless sensor network applications, this function is essential because input is how our SuperNode receives data from com ports.

Variable names

In uBASIC, variable names are stored as a single character, this leads to inconvenience since there are only 26 possible variables and these variable names can only be one single character. In contrary, ntuBASIC hashes the variable names (string) into a long integer and uses a link-list to store these variables. Therefore, each variable is able to have a name with multiple characters. This improves the readability of the script and supports more than 26 variables at the same time.

Support strings and integers

uBASIC variables supports integer only. If a smart sensor node wants to pass a received packet that contains a string to another node, it will need the ability to buffer the string and further send it. Obviously, integer variables are not enough for wireless sensor network applications. This is the reason string variables were made enabled in our version.

Wireless extension

The interpreter wireless extension contains a set of intuitive syntax for beginners to learn to develop wireless programming rapidly. This extension intends to provide most essential wireless sensor services for domain experts.

By benefitted from the robust underlying networking ability Contiki provides, our interpreter does not need to take routing into consideration in wireless transmissions. Consequently, the application layer regards every single end to end distance one hop away.

The design of this extension is fairly simple. It includes an active sending command and a passive receiving subroutine:

Syntax: `send <data> to <IP address>`
This sends an UDP packet with data content to a specific IP address. Moreover, the received event tag of targeted node will be triggered.
Ex: `send 3 to 172.16.0.3`

Event: `received`
This subroutine is fired whenever a packet arrival occurs.
Ex: `received: print "UDP received!"`

©2014 Engineering and Technology Publishing

Hardware extension

The hardware extension currently supports syntax: port <port> line <line>. This syntax offers programmers to access the hardware pins in a variable manner. For example, "clear port 5 line 4" lights a led on a SuperNode. Note that due to the circuit design, 0 (clear) indicates the led to be on. On the other hand, 1 (set) turns off the led. Another keyword is toggle, which simply turns 0 into 1 and 1 into 0. "Toggle port 5 line 4" either turns the turned-off led on or turns the turned-on led off.

Over-the-air reprogramming

Over-the-air reprogramming (OTA) often refers to the capability to reprogram devices through wireless packets. One major advantage is that OTA makes smart objects programmable from locations within the range of wireless network signal. In our work, because of the Internet networking ability Contiki provides, we are possible to reprogram smart objects from one point of the network to another.

Taking benefits from our interpreter, smart object reprogramming only involves the replacement of the running script. In other words, replacing the running script of the devices would affect their behavior at the same time. Therefore, for our OTA design, we replace the interpreter's script in a straight forward manner. Moreover, since replacing the script does not involve modification of the firmware or the OS, this OTA design may provide efficiency and security simultaneously.

In our design, any received UDP packet is parsed by the interpreter to identify if the current receiving packet is for the interpreter or for the running script. As a result, the first byte of the received UDP packet is reserved to indicate the packet destination. By utilizing this design, the length of the script is not limited to the packet size. Therefore, over the air download can be done easily and successfully.

Integrated Development Environment Design

In order to provide our users an intuitive and convenient development environment, a plugin for an open source editor is provided. Geany is an open source editor that is developed to provide a small and fast IDE. It supports syntax highlighting for a wide variety of programming languages, including BASIC language. Plugins for Geany can be developed by using GTK+. In our work, we developed a plugin that makes remote programming for smart objects achievable. In contrast to traditional developing procedure, our system only requires the user to assign the targeted node address and click download as Fig. 4. Fig. 5 shows the architecture of developed system.

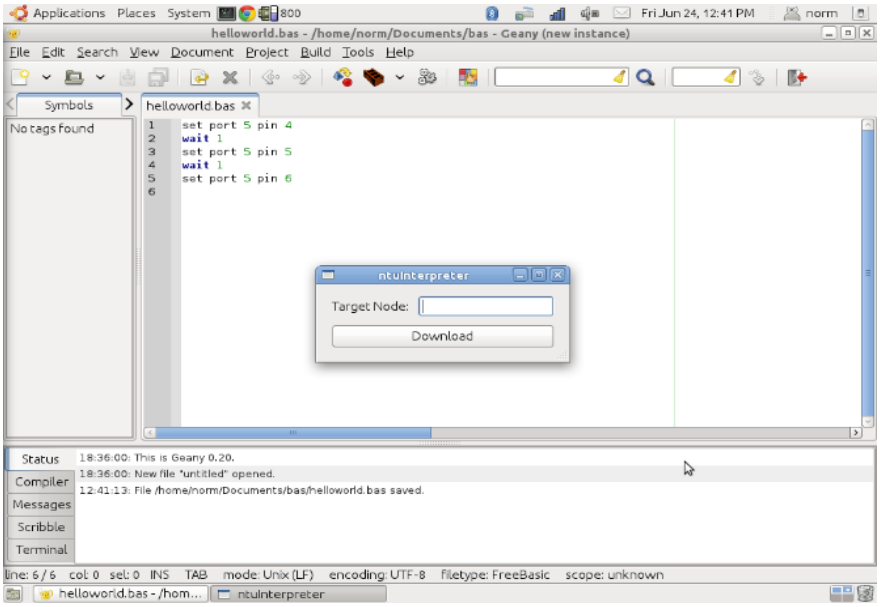


Figure 4. IDE of NTUpreter.

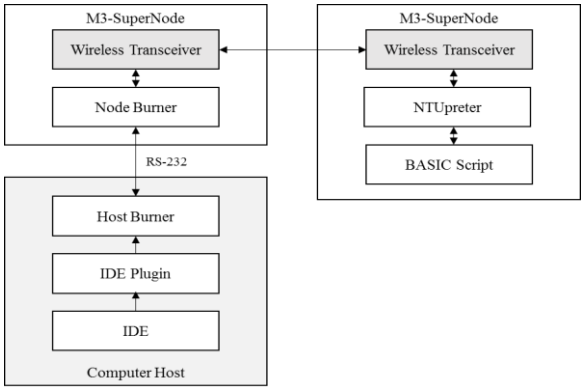


Figure 5. System architecture and relation.

V. Conclusion

In conclusion, in this work, we have focused on to lower the developing difficulties of the smart objects. We allow users from different disciplines to easily learn to program smart objects and make sensor network programming accessible to novice users. With the aim to lower the difficulty of programming smart devices, we have implemented an interpreter to provide structured higher level language programming. This interpreter allows domain experts without programming related background to program smart devices with structured BASIC language and our designed language extensions. Moreover, this work also provides an integrated

©2014 Engineering and Technology Publishing

development environment that help users to download their programs to the interpreter by using wireless packets. The simplicity and convenience we provide in this work offers an environment for domain experts to perform smart objects programming easily from any locations in the sensor network.

ACKNOWLEDGMENT

This study is conducted under the "Advanced Sensing Platform and Green Energy Application Technology Project" of the Institute for Information Industry which is subsidized by the Ministry of Economy Affairs of the Republic of China.

APPENDIX: NTUPRETER SYNTAXES

Data manipulation

let: assigns a value (which may be the result of an expression) to a variable.

```
let x = 10
x = y + 1
```

Program flow control

✓ **if** ... **then** ... **else**: used to perform comparisons or make decisions.

```
if x > 10 then set port 5 line 4 else clear port 5 line 4
```

(multi-line if)

```
if x > 10 then
        set port 5 line 4
else
        clear port 5 line 4
endif
```

✓ **for** ... **to** ... **next**: repeat a section of code a given number of times.
A variable that acts as a counter is available within the loop.

```
for i = 0 to 10
        print i +10
next i
```

✓ **gosub**: temporarily jumps to labeled line, returning to the following line after encountering the **return** command.
This is used to implement subroutines.

```
gosub sayhello
...
sayhello:
        print "hello!"
        return
```

Input and output

✓ **print**: displays a message on the screen (rs232 comport)

```
print "Hello World!"
```

✓ **input**: asks the user to enter the value of a variable. The statement may include a prompt message.

REFERENCES

[1] J. S. Miller, P. A. Dinda, and R. P. Dick, "Evaluating a BASIC approach to sensor network node programming," in *Proc. the 7th ACM Conference on Embedded Networked Sensor Systems*, Berkeley, California, 2009..

[2] P. Levis and D. Culler, "Maté A tiny virtual machine for sensor networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 85-95, 2002.

[3] T. Lindholm and F. Yellin, *Java Virtual Machine Specification*, Addison-Wesley Longman Publishing Co., Inc, 1999.

[4] S. Edwards, *Programming and Customizing the Basic Stamp Computer*, McGraw-Hill, Inc, 1998.

[5] J. K. Juntunen, M. Kuorilehto, M. Kohvakka, V. A. Kaseva, M. Hannikainen, and T. D. Hamalainen, "WSN API: Application programming interface for wireless sensor networks," in *Proc. IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2006, pp. 1-5.

[6] D. Adam, uBASIC. [Online]. http://www.sics.se/~adam/ubasic/.

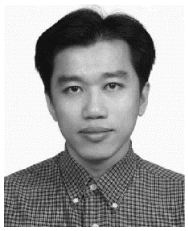
[7] Microsoft, Differences Between GW-BASIC and QBasic, 2003.

[8] Moteiv. (2006). Tmote Sky Datasheet. [Online]. Available: http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf

[9] L. Philip, M. Sam, P. Joseph, S. Robert, W. Alec, G. David, H. Jason, W. Matt, B. Eric, and C. David, *TinyOS: An Operating System for Sensor Networks*, Springer Verlag, ed. 2004.

[10] A. Dunkels. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. pp. 455-462, 2004.

[11] Y. T. Chen, T. C. Chien, and P. H. Chou. "Enix: A lightweight dynamic operating system for tightly constrained wireless sensor platforms," in *Proc. 8th ACM Conference on Embedded Networked Sensor Systems*, Zurich, Switzerland, 2010.



Wen-Jong Wu was born in Taipei, Taiwan, in 1974. He received the B.S degree from the Department of Civil Engineering, National Taiwan University, Taipei, in 1996, and the M.S. and Ph.D. degrees from the Institute of Applied Mechanics, National Taiwan University, in 1998 and 2003, respectively. He is currently an Associate Professor in the Department of Engineering Science and Ocean Engineering, National Taiwan University, where he set up the Opto-Mechantronics Laboratory. His current research interests include system design and integration of precision metrology, smart sensor networks, and piezoelectric power devices.

Yu-Cheng Norm Lien was born in Taipei, Taiwan, in 1985. He received the B.S. degree from the Department of Computer Science, National Chiao Tung University, Hsinchu, in 2008, and the M.S. degree from the Department of Engineering Science and Ocean Engineering, National Taiwan University in 2010. He has been a graduate student working toward the Ph.D. degree in the Department of Engineering Science and Ocean Engineering, National Taiwan University. His current research interests include wireless sensor networks and sensory data compression.

©2014 Engineering and Technology Publishing