Available online at www.sciencedirect.com
SciVerse ScienceDirect
Procedia Computer Science 17 (2013) 579 – 582
Procedia Computer Science

Information Technology and Quantitative Management (ITQM2013)

Porting LooCI Components into Zigduino

Zhun Shen[a], Ka Lok Man[b], Hai-Ning Liang[b], Nan Zhang[b], David Olalekan Afolabi[b] , Eng Gee Lim[b,*]

[a]*IBM, Shanghai, China*
[b]*Xi'an Jiaotong-Liverpool University, Suzhou, China*

Abstract

Loosely-coupled Component Infrastructure (LooCI) is a middleware for building distributed component-based WSN applications. LooCI cleanly separates distributed concerns from component implementation, supports application-level interoperability between heterogeneous WSN platforms, and provides compatibility testing of bindings at runtime.In this paper, we describe our approach to porting LooCI/Contiki from the Raven platform to the Zigduino platform, which is an Arduino-compatible microcontroller environmentthat integrates an 802.15.4 radio on the board.

Keyboards: Loosely-coupled Component Infrastructure; Middleware; Zigduino; Wireless Sensor Networks; AVR

1. Introduction

Wireless Sensor Networks (WSNs) consist of large numbers of tiny sensor devices with wireless communication capabilities, whichcollect information and relay this information to a specific central node. This central node isresponsible for more complex data analysisof the data and also for passing information gathered to other nodes [1].The Loosely-coupled Component Infrastructure (LooCI) is a middleware for building distributed component-based WSN applications. LooCIcleanly separates distributed concerns from component implementation, supports application-level interoperability between heterogeneous WSN platforms, and provides compatibility testing of bindings at runtime[2]. LooCI runs on the Contiki platform, and provides light weight mechanisms and abstractions, that exploit the Contiki OS, which provides a rich execution environment while operatingwithin the limitations of sensor 'motes', which provide microcontrollers running at tens of MIPS, under 20KB of RAM and a few hundred KB flash. Contiki provides dynamic loading and unloading of individual programs and services [1], together with a hybrid interaction model that combines lightweight 'proto-threads' with events. In this research we aim to port LooCI running on the Contiki platform to the Zigduino platform.The rest of this paper is structured as follows.Section 2 provides background and discusses the motivation for this work. Section 3 presents implementation and evaluation. Section 4 discusses our results. Finally Section 5 summarizes and discusses directions for future work.

* Corresponding author. Tel: +86-512-8816-1405; fax: +86-512- 8188-0442.
E-mail address: enggee.lim@xjtlu.edu.cn.

2. Background and Motivation

2.1. Hardware

Arduino is one of the most common hardware platforms. Arduino is small, low cost and modular. The Arduinois used not only for prototyping but also for creating interactive wireless sensor network nodes. Despite their many advantages, previous Arduino boards lacked wireless connectivity, making it impossible to use them in a Wireless Sensor Network (WSN) [4].The Zigduino platform is an Arduino-compatible microcontroller platform that integrates an 802.15.4 radio, which supports 802.15.4-based protocols including: ZigBee, Route Under MAC/6LoWPAN, and RF4CE[7]. It uses a reverse polarity SMA connector (RP-SMA) for an external antenna, which allows the user touse any existing 2.4 GHz antenna. All I/O pins on Zigduino are 5V compatible and can also run at 3.3V. Zigduino is based around ATmega128RFA1, and has 128 KB of flash, on-board memory, of which 2 KB is occupied by the boot-loader. The Zigduino has 16 KB of SRAM (the most of any Arduino-compatible board) and 4 KB of EEPROM, which can be accessed through the EEPROM library[5].The picture below shows a production Zigduino kit with all components.

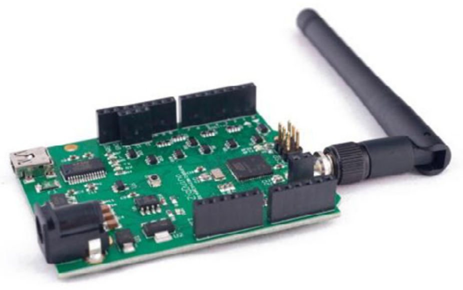
Fig. 1. A picture of the Zigduino components.

2.2. Contiki OS

Contiki is an operating system developed for such constrained environments and provides dynamic loading and unloading of individual programs and services. The kernel is event-driven, but the system supports preemptive multi-threading that can be applied on a per-process basis. Preemptivemulti-threading is implemented as a library that is linked only with programs that explicitly require multi-threading [1]. Contiki is implemented in the C language and has been ported to a number of microcontroller architectures, including the Texas Instrument'sMSP430 and the Atmel AVR.

2.3. A Loosely-coupled Component Infrastructure

Loosely-coupled component infrastructure (LooCI) is composed of a runtime re-configurable component model, a hierarchical type system and a distributed event bus (see Figure 2). LooCI provides a clean separation of distribution concerns from component implementation, which allows components to be re-used in different network environment and supports multiple languages and operating systems. LooCI also provides compatibility testing between component interfaces at bind time. Together, these features promote safe and efficient application development, management and reconfiguration [2]. LooCI runs on the Contiki operation system and is also event-based. LooCI also provides mechanisms for remote reconfiguration, which plays a role in managing application dynamism, which arises from evolving requirements, changing environmental conditions, mobility and unreliable networking [2].

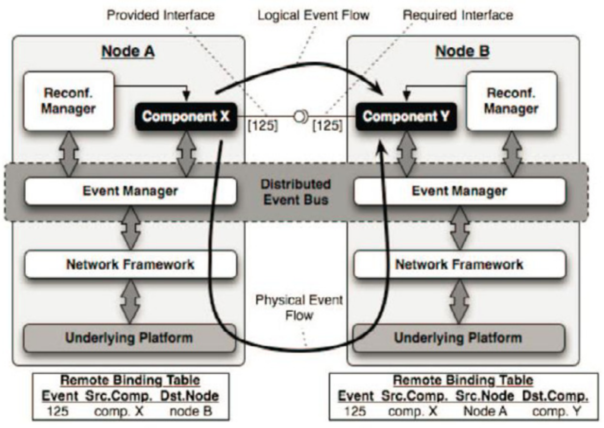

Fig. 2 TheLooCI architecture and bindings.

3. Implement and Evaluation

There are basically two directories which are relevant when porting Contiki to a new platform:(1) /platform and (2) /cpu. In this research, we begin by modifying the LooCI port for the AVR Raven for the Zigduino platform. In this approach we modify the ports of the avr-zigduino (i.e., Atmega128RAF1) to suit our purpose.The CPU specific code is found in /cpu/cpu_name directory. The Makefile from /platform/platform_name is the configuration file of hardware, which usually just specifies some files to be includedand some commands to be executed [6]. This is however the ideal case and in practice, when developing a port, it is usually the case that the Makefile from /platform/platform name configures the paths, hardware and types. When the port is complete one can then try to clean up this Makefile and make it look much more similar to the ideal one. Another Makefile that needs to be considered is theCPU specific one located in /cpu/cpu name. This Makefile looks after the cross-compiler rules and definitions and specifies the path where to look for source files.

In addition, the following files are relevant to porting to the Zigduino platform:

- set-eeprom.c, which is the function file for burning blank Contiki to Zigduino platform.
- contiki-conf.h, which is the hardware configuration file.
- zigd-shell.c, which is the shell function on Zigduino platform.
- Looci.h, which is the component definition file.
- contiki-main.c, which is the system starter with main() function. It also starts most of the Contiki services.
- leds-arch.c, which is the architecture-dependent implementation of LED API of Contiki.
- Makefile, which is a make file which starts compilation of the platform files.
- Makefile, which includes all the other options needed.
- Makefile.target, which is needed to set the target platform.
- Makefile.avr-zigduino, which has the settings needed to compile the avr-zigduino platform.

4. Analysis of Results

The aim of this research is to identify a viable approach to porting the LooCI component model to the Zigduino platform.This presents several challenges and we have tackled them in a sequential manner[6].First, we need to install Contiki on zigduino, which can be found on github[8]. The next challenge is to write the required code to show that LooCI works by causing an LED to blink on Zigduino platform. To overcome this challenge, we have relied on the Contiki 2.5 documentsand have used and adapted both etime.h and process.hto create the new code. The shell function was then also used to check whether the network can run using two Zigduino boards. Another challenge we have encountered in the process is to have a LooCI environment built into the elf file with a blink component.In order to evaluate the LooCI environment,shell is alsoused to start and stop the blink component. The final challenge is to get the programming working for the LooCI's wireless communication. In short, to evaluate the entire system, we have created a blank LooCI self-image installing this image, constructed the blink component, and deployed this component over the air using the shell. At the end finally,the shell is used to start the blink component.

The final outcome is a complete port specific for the LooCI component. A LooCI component containing a "blinking lights" process has successfully been flushed to Zigduino and tested with positive results.

5. Summary and Future Work

In this research, we attempt to find a suitable approach to porting LooCI, a middleware for building distributed component-based wireless sensor network application, to the Zigdruino platform, an Arduino-compatible microcontroller environment that integrates an 802.15.4 radio on the board. We have successfully tested the new LooCIport using the LED blinking application. Beyond Zigdruino, we have shown that with our approach LooCI is portable to different hardware devices.

Our future work will focus on evaluating the performance and efficiency of the LooCI/Zigduino port in comparison to other LooCI ports in terms of energy consumption and the efficiency of component installation, binding and execution.

References

[1] A.Dunkels,B.Gronvall, andT. Voigt, Contiki - a lightweight and flexible operating system for tiny networked sensors,In 29th Annual IEEE International Conference on Local Computer Networks (2004), pp. 455- 462.

[2] D. Hughes, K. Thoelen, J. Maerien, N. Matthys, J. Del Cid, W. Horre, C. Huygens, S. Michiels, and W. Joosen, LooCI: The Loosely-coupled Component Infrastructure, In 11th IEEE International Symposium onNetwork Computing and Applications (NCA'12) (2012), pp.236-243.

[3] W. Horre, D. Hughes, K.L. Man, S. Guan, B. Qian; T. Yu,H. Zhang,Z. Shen,M. Schellekens, and S. Hollands,Eliminating implicit dependencies in component models, IEEE 2nd nternational Conference on Networked Embedded Systems for Enterprise Applications (NESEA'11) (2011), pp.1-6.

[4] V. Georgitzikis, O. Akribopoulos, I. Chatzigiannakis, Controlling Physical Objects via the Internet using the Arduino Platform over 802.15.4 Networks, IEEE Latin America Transactions (Revista IEEE America Latina) (2012), vol.10, no.3, pp.1686-1689.

[5] Logos-electro,Onlien: http://logos-electro.com/zigduino/ [accessed on December 2012].

[6]S. Alexandru, Porting the Core of the Contiki, (2007), Online: http://www.eecs.iu-bremen.de/archive/bsc-2007/stan.pdf [accessed on December 2012].

[7] Logos-electro,Online:http://logos-electro.com/zigduino/ [accessed on December 2013].

[8] Maniacbug, Online:https://github.com/maniacbug/contiki-avr-zigduino/wiki/Installing [accessed on December 2013]