Modelling and Analysis of LooCI Models in Zigduino

David Olalekan Afolabi, Zhun Shen, Ka Lok Man*, Hai-Ning Liang, Nan Zhang and Eng Gee Lim

Abstract—**The Zigduino is an open-source Arduino compatible microcontroller platform with an integrated 802.15.4 radio. The Loosely-coupled Component Infrastructure (LooCI) is a component-based middleware for building sensor network applications that runs on the Contiki operating system, which provides IPv6 networking. In this paper, we describe our approach to, and experiences of porting the LooCI/Contiki stack to the Zigduino platform.**

***Index Terms*—LooCI; Contiki; Zigduino; WSN; AVR**

I. Introduction

Wireless Sensor Networks (WSN) consist of large numbers of tiny sensor devices with wireless communication capabilities which gather sensor data on the physical environment and transmit this data to more powerful servers for analysis [1]. Loosely-coupled Component Infrastructure (Loo-CI) is a middleware for building distributed component-based WSN applications. LooCI cleanly separates distribution concerns from component implementation, supporting application-level interoperability between heterogeneous WSN platforms, and provides a rich type system [2].

LooCI runs on the Contiki operating system that provides a specialized set of abstractions that can be used to build highly efficient embedded software. Specifically, Contiki provides dynamic loading and unloading of individual programs and services [1]. In this paper we report on our experiences of porting LooCI and Contiki platform to the Zigduino platform.

LooCI is comprised of a runtime reconfigurable application level component model, a hierarchical type system and a distributed event bus. Its features promote safe and efficient application development, management and reconfiguration [2] Zigduino platform is an open sources Arduino-compatible microcontroller platform that addresses this problem by integrating an 802.15.4 radio, it has powerful wireless communication ability. LooCI is programmed by C and codes are host by Google Code, Zigduino and Contiki have good programming support. So we plan to port LooCI on Zigduino in order to widely spread LooCI on more open source hardware in WSN.

The remainder of this paper is structured as follows. Section II provides background and discusses the motivation for this work. Section III presents implementation and evaluation. Section IV discusses our results. Finally Section V summarizes and discusses directions for future work.

II. BACKGROUND AND MOTIVATION

A. Hardware

Arduino is one of the most common hardware platforms because of its small size, low cost and modularity; it is used not only for prototyping but also for creating interactive applications. Despite its many advantages, the basic Arduino platform lacks wireless connectivity, which makes it unsuitable for supporting WSN applications [4]. The Zigduino platform is an Arduino-compatible microcontroller platform that addresses this problem by integrating an 802.15.4 radio. The Zigduino offers a reverse polarity SMA connector (RP-SMA) for an external antenna. All I/O pins on Zigduino are 5V compatible and can also runs at 3.3V. Zigduino is based around ATmega128RFA1, and has 128 KB of flash memory of which 2 KB is occupied by the boot loader. It also has 16 KB of SRAM 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.

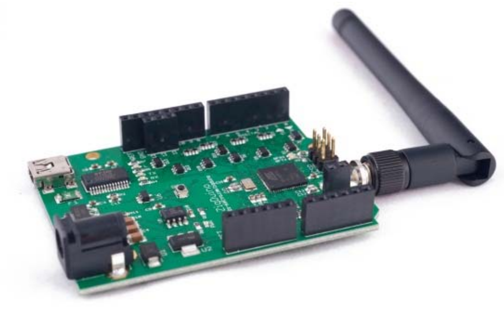

Fig. 1. A picture of the Zigduino components

Manuscript received December 25, 2012; revised January 30, 2013.

David Olalekan Afolabi is with the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China (e-mail: david.afolabi09@student.xjtlu.edu.cn).
Zhun Shen is with IBM, Suzhou, China (e-mail: shenzhun@outlook.com)
Ka Lok Man is with the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China (e-mail: ka.man@xjtlu.edu.cn).
Hai-Ning Liang is with the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China (e-mail: haining.liang@xjtlu.edu.cn).
Nan Zhang is with the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China (e-mail: nan.zhang@xjtlu.edu.cn).
Eng Gee Lim is with the Department of Electrical and Electronic Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China (e-mail: enggee.lim@xjtlu.edu.cn).
* Corresponding author

B. Contiki OS

Contiki OS is designed to satisfy the need for lightweight mechanisms and abstractions that provide a rich enough execution environment while staying within the limitations of the constrained devices [1]. Typical sensor devices are equipped with 8-bit microcontrollers, code memory on the order of 100 kilobytes, and less than 20 kilobytes of RAM [1]. Contiki provides dynamic loading and unloading of individual programs and services that is used by LooCI to support Over The Air (OTA) component deployment. The kernel is event-driven, but the system supports preemptive multi-threading that can be applied on a per-process basis. Preemptive multi-threading is implemented as a library that is linked only with programs that explicitly require multi-threading [1]. This threading approach is used by LooCI to host multiple concurrently executing components. Contiki is implemented in the C language and has been ported to a number of microcontroller architectures, including the Atmel AVR, which is used on the Zigduino.

C. A Loosely-coupled Component Infrastructure

The 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. LooCI also supports multiple languages and operating systems. Together, these features promote efficient application development, management and reconfiguration [2]. In addition, LooCI plays a role in managing application dynamism, which arises from evolving requirements, changing environmental conditions, mobility and unreliable networking [2].

III. IMPLEMENT AND EVALUATION

According to existing work, LooCI runs on AVR Raven. The aim of this research is to identify a viable approach to port the LooCI component to the Zigduino platform. It presents several challenges and we have tackled them in a sequential manner [6]. First, we need to install Contiki on Zigduino, which was achieved using the Contiki port for Zigduino that is available on Github. The next challenge was to write the required code to show LEDs blinking on Zigduino platform. To overcome this challenge, we have relied on the Contiki 2.5 doc and have used and adapted both etime.h and process.h to create the new code. Then the shell function is 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 running well on Zigduino, shell modules in Contiki OS is also used 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 elf version LooCI image and deploy blank image on Contiki on Zigduino , then construct the blink component, and deployed this component over the air using the shell. At the end, the shell modules are used to start the blink component in order to check that LooCI component is successfully deployed on Zigduino.

The final outcome is a complete port specific for the LooCI component. All functionalities of LooCI have been exhaustively tested through the help of already existing applications and by writing new ones.

A LooCI component containing a "blinking lights" process has successfully been flushed to Zigduino and tested with positive results. In order to exemplify the architecture of LooCI and how to write application on LooCI, "Blinking Lights" component is listed as an example.

1) Code

```
#include "looci.h"
#ifdef LOOCI_BLINK_DEBUG
#include <stdio.h>
#endif
#ifdef BUILD_COMPONENT
#undef PRINTF
#define PRINTF(...)
#endif

COMPONENT(blink, "LED Blink");
AUTOSTART_COMPONENTS(&blink);
COMPONENT_THREAD(blink, ev, data)
{
 COMPONENT_BEGIN();
 static struct etimer et;
 while(1) {
 etimer_set(&et, CLOCK_SECOND * 2);
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
}
  COMPONENT_END();
}
```

2) A code walk-trough

Setting component up, requires that the Contiki and LooCI header files define the following information.

```
#include "looci.h"
#ifdef LOOCI_BLINK_DEBUG
#include <stdio.h>
#endif
#ifdef BUILD_COMPONENT
#undef PRINTF
#define PRINTF(...)
#endif
```

Declaring the blink component itself and its human-readable component-type:

```
COMPONENT(blink, "LED Blink");
```

General structure of a component has four macros, `AUTOSTART_COMPONENT` is used to run the blink component automatically. `COMPONENT_THREAD(blink, ev, data)` macro is the main method to run components, the first parameter `blink` is the name of the variable holding the component metadata as declared above; The second argument `ev` is the low-level Contiki event type that caused the component execution to be scheduled and the third argument `data` is a pointer to extra data passed by the Contiki kernel. `COMPONENT_START` is used to start the component. `COMPONENT_END` is used to stop the component and clean the running space of this component.

`Etimer_set(…)` is a timer running every 2 times system clock. `PROCESS_WAIT_EVENT_UNTIL()`wakes us up using the timer.

```
AUTOSTART_COMPONENTS(&blink);
COMPONENT_THREAD(blink, ev, data)
{
COMPONENT_BEGIN();
 static struct etimer et;
 while(1) {
 etimer_set(&et, CLOCK_SECOND * 2);
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    }
COMPONENT_END();
   }
```

IV. SUMMARY AND FUTURE WORK

In this research, we attempt to find a suitable approach to port LooCI, a middleware for building distributed component-based wireless sensor network application, into the Zigdruino platform, an Arduino-compatible microcontroller environment that integrates an 802.15.4 radio on the board. In this paper, we describe our approach to migrate LooCI / Contiki running on the Raven platform to the Zigduino platform. These experiences show that it is possible to quickly port the LooCI component model to new platforms.

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.

ACKNOWLEDGEMENTS

The authors are pleased to acknowledge support from the iMinds-DistriNet research group of KU Leuven, Belgium, the authors of the original LooCI platform. In addition to this, this work was partially supported by the Xi'an Jiaotong-Liverpool University (China) Research Development Fund under Grants RDF10-01-27 and RDF10-02-03.

REFERENCES

[1] A. Dunkels, B. Gronvall, and T. 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 on Network 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].