Porting LooCI from the Contiki Platform to the Zigduino Platform: An Working Approach

Zhun Shen, David Olalekan Afolabi, Hai-Ning Liang, Nan Zhang, Dawei Liu, Ka Lok Man[*], Eng Gee Lim, Chi-Un Lei, Yue Yang, Lixin Cheng

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 (LooCI) [21] is a middleware for building distributed component-based WSN applications. In contrast to the loose-coupling feature offered by LooCI, tightly-coupled component models for networked embedded systems are found in NesC [7], OpenCOM [8], RUNES [9] and REMORA [10], just to name a few. Remote Procedure Call (RPC) is often adopted as the means for interaction between the sensors by the above-mentioned models. However, this RPC-based interaction is a bad fit with the characteristics of WSN for three key reasons:

1. *Unreliable networking*: the low-power wireless networking protocols available to WSN motes are inherently unreliable, whereas RPC-calls require a reliable connection between cooperating components. While this could be addressed by implementing reliable networking on top of WSN protocols, this has a high cost in terms of power-consumption [11].

2. *Tight coupling*: the use of RPC results in a tight coupling between cooperating components, which has two negative effects. Firstly, the failure of one node may results in the failure of another. Secondly, to safely enact reconfiguration between nodes cooperating using RPC it is necessary to place both nodes into quiescent state [12]. This is problematic as contemporary quiescence protocols do not scale well in unreliable network environments.

3. *Centralized architecture*: while it is not inherent to the RPC model, current implementations of RPC [13] use a centralized service registry to mediate the discovery of software services. As has been argued in [14], this result in an implicit distributed dependency, which is invisible to the developer and thus cannot be reasoned about.

To address the above problems, LooCI was designed and it better fits the dynamic aspect of WSNs for the following reasons:

1. *Loose-coupling*: cooperating components is loosely coupled. This ensures that (a) the failure of a component does not lead to the failure of remote component and (b) that components may be reconfigured without the need for expensive distributed quiescence protocols.

2. *Decentralization*: the LooCI interaction model for WSN is decentralized so that the failure of a centralized system element, or intermittent connectivity do not lead to the failure of cooperating motes.

3. *Separation of concerns*: as described previously, component models facilitate a separation of concerns, wherein embedded developers provide reusable units of software functionality and application developers compose these components together to form applications. A WSN binding model should therefore cleanly separate distributed concerns from local software implementations.

LooCI is considered an important contribution to future WSNs because of its simplicity in implementation, small initialization overhead, and low memory requirement. WSNs are characterized by their high heterogeneity. Take environment monitoring as an example. Commonly used environment monitoring sensors include temperature sensors, CO/CO2 sensors, and noise sensors. These sensors can differ from each other in the operating systems and in the underlying hardware, which pose a barrier for

Zhun Shen is with IBM (Suzhou), China (e-mail: shenzhun@outlook.com).
David Olalekan Afolabi, Hai-Ning Liang, Nan Zhang, Dawei Liu, Ka Lok Man are with the Department of Computer Science and Software Engineering, while Eng Gee Lim is with the Department of Electrical and Electronic Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China (e-mail: david.afolabi09@student.xjtlu.edu.cn and {haining.liang, nan.zhang, dawei.liu, ka.man, enggee.lim}@xjtlu.edu.cn).
Ka Lok Man is also with the Baltic Institute of Advanced Technologies, Lithuania.
Chi-Un Lei is with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, China (e-mail: culei@eee.hku.hk).
Yue Yang and Lixin Cheng are with the Suzhou Institute of Nano-Tech and Nano-Bionics, Chinese Academy of Sciences, China (e-mail: yangyue@hotmail.com, clx009@gmail.com).
* Corresponding author: Ka Lok Man (ka.man@xjtlu.edu.cn).

interoperability. LooCI can help to remove the barrier and support the interoperation between programs written in different languages. Small initialization overhead implies short initialization time. This is very important for real-time applications like target tracking and object localization. In order to minimize the energy consumption, sensor nodes are commonly put in sleep mode until a target is identified or an object is arriving. For those sensor nodes in sleep mode, it is critical to be able to wake up immediately. LooCI can be initialized within a few milliseconds, making it suitable for real-time applications. Finally, compared with traditional operation systems of WSNs, such as TinyOS, the memory cost of LooCI is much smaller. This makes it suitable for the sensor nodes that have very limited memory space.

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 lightweight execution environment, runtime reconfigurable component model and an event-based binding model. Its features promote safe and efficient application development, management and reconfiguration [2]. LooCI Reconfiguration Engine maintains references to all local components and enacts incoming reconfiguration commands that are received over the event bus. As all reconfiguration occurs over the event bus, it is possible for any component to enact reconfiguration of any other component within the network, subject to access control policies. The LooCI component model is platform and language agnostic, allowing developers to implement components in various languages and for different operating systems. Upon deployment, a LooCI component registers with the local Reconfiguration Engine, which supports introspection of component state and lifecycle control. The LooCI event-bus is an asynchronous, event based communication medium that promotes loosely coupled interactions. On the one hand, synchronization decoupling is provided by non-blocking interactions between components and the event bus. On the other hand, loose coupling in space is realized by separating distribution concerns from component implementation.

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.

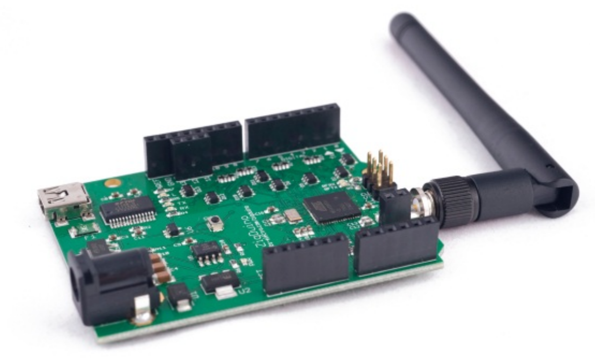

Figure 1. A picture of the Zigduino components.

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].

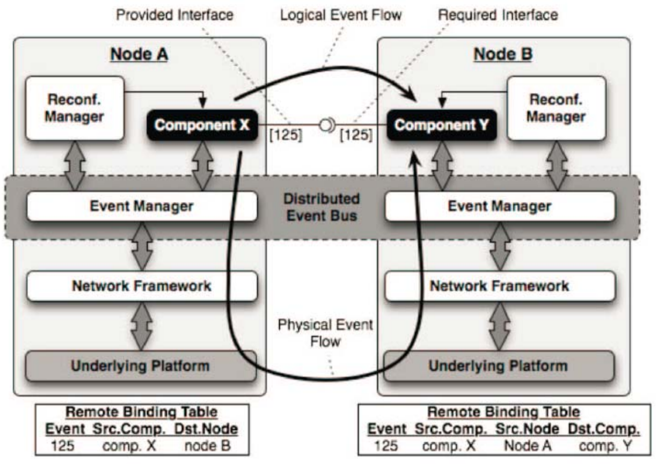

Figure 2. LooCI architecture and bindings.

III. IMPLEMENT AND EVALUATION

According to existing work, LooCI runs on AVR Raven (see Figure 3), a board which supports wireless transmissions.

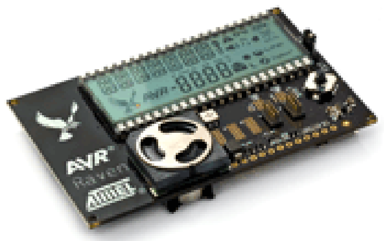

Figure 3. The AVR Raven architecture.

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, as shown in Figure 4.

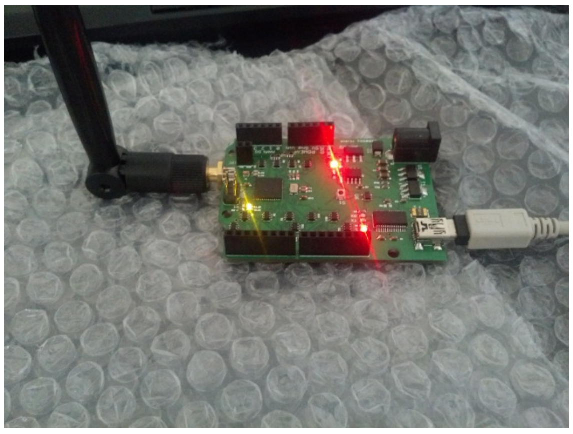

Figure 4. LED lights blinking on the Zigduino board.

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 following code:

```
#ifndef __ETIMER_H__
#define __ETIMER_H__
#include "sys/timer.h"
#include "sys/process.h"
struct etimer {
struct timer timer;
struct etimer *next;
struct process *p;
};
CCIF void etimer_set(struct etimer *et, clock_time_t
interval);
CCIF void etimer_reset(struct etimer *et);
void etimer_restart(struct etimer *et);
clock_time_t etimer_expiration_time(struct etimer *et);
clock_time_t etimer_start_time(struct etimer *et);
CCIF int etimer_expired(struct etimer *et);
void etimer_stop(struct etimer *et);
PROCESS_NAME(etimer_process);
#endif /* __ETIMER_H__ */


blink.c
#include "contiki.h"
#include <stdio.h>
#include <avr/io.h>
#include "contiki-conf.h"
#include "dev/leds.h"
#include "sys/etimer.h"
#include "sys/process.h"

/*
 * PORT where LEDs are connected
 */
#define LED_PORT0                        (PORTB)
#define LED_PORT_DIR0                    (DDRB)
#define LED_PORT                         (PORTD)
#define LED_PORT_DIR                     (DDRD)

/*
 * PINs where LEDs are connected
 */
#define LED_PIN_0                        (PB1)
#define LED_PIN_1                        (PD5)
#define LED_PIN_2                        (PD6)

static int count=0;
static struct etimer et;

PROCESS(blink_process,"LED blinks");
AUTOSTART_PROCESSES(&blink_process);

PROCESS_THREAD(blink_process,ev,data)
{
PROCESS_BEGIN();

void
leds_arch_init(void)
{
  LED_PORT0 |= (1 << LED_PIN_0);
  LED_PORT_DIR0 |= (1 << LED_PIN_0);
  LED_PORT |= (1 << LED_PIN_1);
  LED_PORT_DIR |= (1 << LED_PIN_1);
  LED_PORT |= (1 << LED_PIN_2);
  LED_PORT_DIR |= (1 << LED_PIN_2);
}

void
leds_arch_off(void)
{
  LED_PORT0 &= ~(1 << LED_PIN_0);
  LED_PORT_DIR0 |= (1 << LED_PIN_0);
  LED_PORT &= ~(1 << LED_PIN_1);
  LED_PORT_DIR |= (1 << LED_PIN_1);
  LED_PORT &= ~(1 << LED_PIN_2);
  LED_PORT_DIR |= (1 << LED_PIN_2);
}

while(1)
{
printf("----count %d'sloop begin----\n",count);
leds_arch_off();
etimer_set(&et,CLOCK_SECOND*2);


printf("----etimer_set begin----\n");

printf("----etimer_expired            =            %d----
\n",etimer_expired(&et));
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
printf("----PROCESS_WAIT_EVENT_UNTIL begin----\n");

leds_arch_init();
printf("----LED Light----\n");
count++;


if(count == 10){
break;
}
}
PROCESS_END();}


code to deply blink on zigduino
#! /bin/bash
cd
contiki-avr-zigduino/platform/avr-zigduino/tools/set-eepro
m
make NODE=3 AVRDUDE_PORT=/dev/ttyUSB0
cd ../../tests/blink
make upload
```

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();
}
```

The following code shows that we can show hello-world sent from one board to another (as shown in Figure 5).

```
#include "contiki.h"
#include "looci.h"

// For testing
#include "reconfiguration/wiring_private.h"

#include <stdio.h>

#include <stdio.h>
#ifdef CONTIKI_TARGET_AVR_RAVEN
#include <avr/pgmspace.h>
#define                             PRINTF(FORMAT,args...)
printf_P(PSTR(FORMAT),##args)
#else
#define PRINTF printf
#endif // CONTIKI_TARGET_AVR_RAVEN

// printf alike things are currently broken for loadable
components.
// Your mileage may vary, so you can try to uncomment this,
but don't
// rely on it working.
#ifdef BUILD_COMPONENT
#undef PRINTF
#define PRINTF(...)
#endif

COMPONENT(sender, "Sender component");
COMPONENT(receiver, "Receiver component");

AUTOSTART_COMPONENTS(&sender, &receiver);

static struct etimer et;

COMPONENT_THREAD(sender, ev, data)
{
  COMPONENT_BEGIN();

  PRINTF("Sender started\n");

  while(1) {
    etimer_set(&et, 5 * CLOCK_SECOND);

    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

    looci_event_publish(100, "Hello world!", 13);
    PRINTF("Sender: event published\n");
  }

  COMPONENT_END();
}

static struct looci_event * recv_event;

COMPONENT_THREAD(receiver, ev, data)
{
  COMPONENT_BEGIN();

  wire_local_all(100, receiver.id);

  while(1) {
    PRINTF("Receiver: waiting for event\n");
    LOOCI_EVENT_RECEIVE(recv_event);
    PRINTF("%s\n", (char*)recv_event->payload);
  }

  COMPONENT_END();
}
```

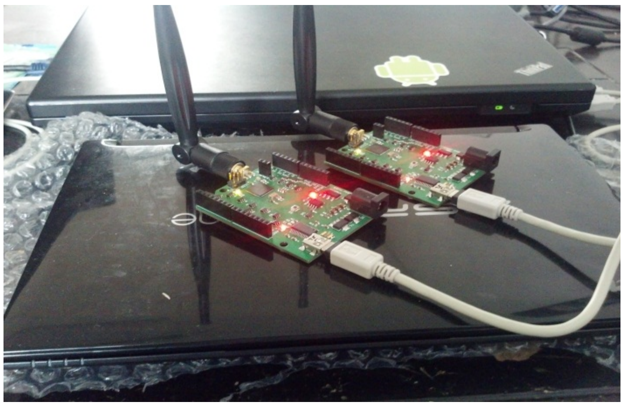
Figure 5. Sending messages between two boards.

IV. Discussion: LooCI and Harvesting Energy from the Environment

The challenge of sustaining long-term operation of wireless senor systems has been investigated in the past decades. Recently, harvesting electrical power from environmental energy sources has become an increasingly feasible option for wireless sensor networks that need to operate autonomously for long periods of time. Micro-scale energy harvesting is a technique that enables electrical energy conversion from other energy forms such as solar, wind, vibration, thermal, electromagnetic energy [15,16,17]. Up to date, there have been some design prototypes of wireless sensor nodes that have been tested and demonstrate the benefits of micro-scale energy harvesting. For example, the authors of [18] demonstrated successful operation of a micro power application based on environmental vibration energy harvesting. The authors of [19] validated the feasibility of using indoor light energy to power a wireless sensor network. The aforementioned testing prototypes and demonstrations show the great potential to use energy harvesting technique to prolong the lifetime of wireless sensor systems.

Designing highly efficient micro-scale energy harvesting systems requires an in-depth understanding of various design considerations and tradeoffs. In [20], the authors provide an overview of the various challenges and considerations involved from maximum power point considerations for micro-scale energy harvesting powered systems. However, most of the research in the literature focuses on hardware system design and implementation for energy harvesting powered wireless sensor systems. Since software operation and hardware implementation is tightly coupled in these systems, it is highly attractive to co-design both aspects of hardware circuit and software platform.

As mentioned earlier LooCI is very simple in its implementation, has a small initialization overhead, and low memory requirements. These features allow it to be a suitable candidate to be powered by harvesting the power from environmental energy sources.

V. 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.

In addition, multi-layer software and hardware adaptivity will be explored to achieve efficient computing and communication. We will integrate various energy harvesting techniques with our LooCI platform, and then adaptively adjust the hardware configuration and software operation of our LooCI system. Through this cross-layer joint optimization methodology, the most energy efficient wireless sensor network application will have a great potential to be created.

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 Xi'an Jiaotong-Liverpool University (Suzhou, China) Research Development Fund under Grants RDF10-01-27 and RDF10-02-03; and Transcend Epoch International Co., Ltd., Hong Kong.

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].

[7] Gay D., Levis P., Von Behren R., Welsh M., Brewer E., Culler D., The NesC Language: A Holistic Approach to Networked Embedded Systems, in Proc. of the conference on Programming Language Design and Implementation, ACM SIGPLAN 2003, San Diego, California, USA, pp. 1 – 11.

[8] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama and T. Sivaharan T, "A generic component model for building systems software", in ACM Transactions on Computer Systems, Vol. 26, No. 1, pp. 1-42, 2008.

[9] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G.P. Picco, T. Sivaharan, N. Weerasinghe and S. Zachariadis, The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario, in proc. of 5th IEEE conference on Pervasive Computing (PerCom'07), White Plains, NY, pp. 69–78, , Mar. 19th-23rd 2007.

[10] Rellermeyer J., Alonso G., Concierge: A Service Platform for Resource-Constrained Devices, in ACM SIGOPS Operating Systems Review, Vol. 41, No. 3, June 2007, pp. 245 – 258

[11] P. Grace, G. Coulson, G.S. Blair, B. Porter and D. Hughes, "Dynamic Reconfiguration in Sensor Middleware", in proc. of 1st International Workshop on Middleware for Sensor Networks (MidSens '06), Melbourne, Australia, pp. 1-6, Nov. 28th 2006.

[12] Grace P., Hughes D., Porter B., Blair G., Coulson G., Taiani F., xperiences with Open Overlays: A Middleware Approach to Network eterogeneity, in Proc. of the European Conference on Computer ystems (EuroSys' 08), Glasgow, Scotland, UK, March 2008, pp. 23-136.

[13] Java Remote Method Invocation (RMI), available online at: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, (accessed 22/09/11).

[14] W. Horré, D. Hughes, K. Man, S. Guan, B. Qian, T. Yu, H. Zhang, Z. Shen, M. Schellekens and S. Hollands, Eliminating Implicit Dependencies in Component Models, in proc. of 2nd IEEE conference on Networked Embedded Systems for Enterprise Applications (NESEA'11), Perth, Australia, Dec. 2011, pp. 1-6.

[15] C. Lu, V. Raghunathan, K. Roy, Micro-scale energy harvesting: a system design perspective, Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 89-94, 2010.

[16] C. Lu, V. Raghunathan, K. Roy, Efficient design of micro-scale energy harvesting systems, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, Vol.1, issue 3, pp. 254-266, 2011.

[17] S. Sudevalayam, P. Kulkarni, Energy harvesting sensor nodes: survey and implications, IEEE Communications Surveys & Tutorials, vol. 13, no. 3, pp. 443-461, 2011.

[18] C. Lu, C.-Y. Tsui, W.-H. Ki, Vibration energy scavenging system with maximum power tracking for micro power applications, IEEE Transactions on VLSI Systems, vol. 19, issue 11, pp. 2109-2119, Nov. 2011

[19] Q. Huang, C. Lu, M. Shaurette, R.F. Cox, Environmental thermal energy scavenging powered wireless sensor network for building monitoring, 28th International Symposium on Automation and Robotics in Construction, pp. 1376-1380, 2011.

[20] C. Lu, V. Raghunathan, K. Roy, Maximum power point considerations in micro-scale solar energy harvesting systems, IEEE International Symposium on Circuits and Systems (ISCAS), pp. 273-276, 2010.

[21] David Olalekan Afolabi, Zhun Shen, Ka Lok Man, Hai-Ning Liang, Nan Zhang, Eng Gee Lim, Modelling and Analysis of LooCI Models in Zigduino, Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists 2013, 13-15 March, 2013, Hong Kong, pp713-715.