An Event-based Component Model for Wireless Sensor Networks: a Case Study for River Monitoring

Jó Ueyama[1], Daniel Roy Hughes[1,2,3], Nelson Matthys[3], Wouter Horré[3], Wouter Joosen[3], Christophe Huygens[3], Sam Michiels[3]

[1]Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brazil

[2]Computer Science and Software Engineering,
Xi'an Jiaotong-Liverpool University
Suzhou Industrial Park, Suzhou, Jiangsu, 215123, China

[3]Department of Computer Science, Katholieke Universiteit Leuven,
Leuven, 3000, Belgium

joueyama@icmc.usp.br, Daniel.Hughes@xjtlu.edu.cn

{nelson.mathys, wouter.horre}@cs.kuleuven.be

{wouter.joosen, christophe.huygens, sam.michiels}@cs.kuleuven.be

Abstract. *This paper examines an event-based component model for wireless sensor networks called LooCI (Loosely-coupled Component Infrastructure) and discusses an application of LooCI for river monitoring in creeks that flow through São Carlos. Our sensor network application which is based on LooCI monitors flood, pollution and human tampering and warns potential stakeholders whenever they are at risk (e.g. of floods). Our pollution monitoring system is used to inspect rivers and issues a warning whenever the condition of water reaches an unacceptable level. In addition, our depth sensor monitors rivers and is used to predict potential floodings. Finally, our three dimensional accelerometer detects any improper vibration caused by tampering with the deployed node. Our component model for sensor networking environments has a novel loosely-coupled binding model which is particularly designed to support intermittent connections between the components. This reflects the nature of wireless sensor networks that are often subject to unreliable networking.*

1. Introduction

Wireless sensor networks (hereafter, WSNs) are composed of tiny embedded computers known as "motes". A mote will have an embedded CPU, low power wireless networking and simple sensors. WSN is been increasingly employed in a large number of applications including habitat monitoring, flood prediction, disaster management, military intelligence and surveillance. As they are normally resource constrained devices, WSNs often require lightweight and reconfigurable platforms. This is because reconfigurable platforms often allow systems to deploy only those functionalities that are required. There are numerous lightweight platforms to program WSNs such as OpenCom [2], RUNES [1], NesC [3]. The main problem with them is that they have a steep learning curve and each of them

relies on a different approach (e.g. component-based, active-messages, etc.). In addition, most of them are based on a RPC (Remote Procedure Call) type of binding, which is not suited to the WSN software domain as they do not reflect the dynamic features of WSN environments. This is because connections in WSN environments are often intermittent and thus, RPC-based bindings do not scale well with them; as well as this, they may require developers to deal with a complex fault-tolerant RPC-based binding.

This paper examines a component model tool (LooCI – Loosely Coupled Component Infrastructure) [4] for WSNs and also explores a case study which is implemented to illustrate river monitoring in creeks that flows through São Carlos – SP. The aim is to give a brief description of LooCI and show the implementation of LooCI in river monitoring on SunSPOT sensor platforms[1]. Both implementations (i.e. of the component model and the case study) can be downloaded at `http://code.google.com/p/looci/`. The blog at `http://sp-river.blogspot.com/` provides rich material for our case study work.

The remainder of this paper is structured as follows: Section 2 summarizes the existing component models and outlines the benefits of our tool. Subsequently, we give details of our LooCI component model for WSNs in Section 3. Following this, Section 4 details an application of LooCI to a WSN-based river monitoring system in creeks running through São Carlos (São Paulo). Finally, Section 6 wraps up this paper with some concluding remarks.

2. Existing Component Models for WSNs and their Limitations

Run-time reconfigurable component models address many critical problems encountered in WSN environments, which have limited resources, are highly dynamic and, as they are expected to operate unattended for long periods of time, must evolve to meet changing application requirements. In our view, component-based approach can reduce these problems, as they can naturally deal with:

- *Dynamism*. Sensor networks are highly dynamic. Component-based reconfiguration offers adaptation mechanisms to handle this dynamism (e.g. by only deploying required functionalities at runtime).
- *Evolution*. As application requirements change over time, component based approaches allow systems to evolve through the deployment of new components.

The Loosely-coupled Component Infrastructure (LooCI) [4] is designed to support embedded Java ME (micro edition) platforms such as the SunSPOT or Java ME smartphones. LooCI comprises an easy-to-use component model and a simple yet extensible networking framework. Each LooCI node is connected via a common event-bus communication substrate. Like other embedded component platforms, such as RUNES [1] or OpenCOM [2], LooCI components support run-time reconfiguration, concrete interface definitions, introspection and support for the re-wiring of bindings.

Unlike OpenCOM or RUNES, LooCI components are indirectly bound over the event bus. Thus, all LooCI components define their interfaces as the set of LooCI events that they publish. The receptacles of a LooCI component can be defined in a similar way to the events which they subscribe. Each LooCI event has a globally unique identifier

[1] www.sunspotworld.com

which classifies the event type in terms of a global descriptive hierarchy. In our view, a loosely-bound component model such as LooCI is an excellent fit with adaptive resource management. As the LooCI binding model is inherently indirect, operating over an event bus, it is possible for a service manager to modify bindings based upon execution context in a manner that is transparent to the upper layers. The fact that binding modification is transparent, makes it easier for us to separate the concerns of resource management and application functionality. This allows end-users to specify the services they require and WSN administrators to tailor system behavior without the need to implement new components. The LooCI event bus also provides a common point of interception to support the gathering of contextual data.

3. The LooCI Component-based Programming Tool

Further details are given here of the implementation aspects of LooCI showing how LooCI developers deal with the event model (e.g. creating events). We also provide an overview of the component model and component types that are supported by LooCI in Section 3.2. It should be noted that the implementation described in this section is particularly designed to run on SunSPOT sensor platforms. However, given that LooCI is a generic form of technology, it can be adopted to run on a variety of platforms. This implementation is compatible with SunSPOT SDK v5.0 Red (090706).

3.1. The Event System

Events are created in two different ways, either by a component wanting to publish an event through the Event Manager or by the Event Manager itself when it receives an event via the radio. In the latter case, the Event Manager recreates the event from the message's payload and dispatches it to the subscribed components. This is handled by the Event Manager and should not be a concern of the application developer. In the former case however, the application developer has to create the event himself. This can be done in the following manner.

```
PayloadBuilder payload = new PayloadBuilder();
payload.addInteger(lightSensor.getAverageValue());
Event event = new Event(EventTypes.LIGHT_READING,
                        payload.getPayload());
publishEvent(event);
```

The PayloadBuilder class is a utility class used to create event payloads. The construction of an event payload is as follows:

# of elements	length of element 1	...	length of element n	element 1	...	element n

An element can be a String, an integer or a byte and these can be stored in the payload by using the appropriate add-methods. The PayloadBuilder is responsible for filling in the specific fields of the payload.

Once the payload is created, an event can be instantiated. This is done by calling the following Event constructor:

```
public Event(byte type, byte[] payload) { ... }
```

The first parameter defines the event type and can be chosen from the EventTypes class. The second parameter is the payload that was previously created. These two parameters are the only values which go on the wire when an event is transmitted. However, the following additional flags can be set for events by a user.

- critical: this declares that the event should be transmitted by using a reliable transport protocol
- remote address: this represents the address of the specific remote node which this event should be sent to.

As these flags are optional, the respective setters have to be explicitly called on the event to set them.

The event system creates a distributed "Event Bus" to which all the LooCI components are connected. A per-node instance of the LooCI Event Manager implements a simple topic-based publish-subscribe event model wherein events are disseminated to subscribers based upon their type. For example, a software element may subscribe to events of type "TEMPERATURE" (created by the above-mentioned operations) and may then be wired to a component at a given network location (local, a remote node or a remote group of nodes) that produces these events. The interface to the event bus is simple and lightweight, however, when combined with the LooCIs global, hierarchical type system, the event manager allows for rich modeling of interactions between the nodes.

3.2. The LooCI Component Model

Two types of components are implemented in the SunSPOT version of LooCI, both relying heavily on the Isolates provided by the SunSPOT SDK. Isolates are process-like units of computation which are isolated from other isolates. The two types of components are macro- and microcomponents.

- Macrocomponents each run in a separate child isolate and communicate with the LooCI runtime middleware in the master isolate via II-RPC (Inter Isolate-Remote Procedure Call). They can execute multiple threads and use utility libraries.
- Microcomponents are finer-grained components which run alongside the LooCI runtime in the master isolate. They must be single threaded and self-contained, and not use any utility libraries.

Once they are deployed and initialized, both macro and microcomponents are controlled (started, stopped, put into quiescent mode) by the Reconfiguration Engine. They can be in 3 states:

- 0 = stopped. The component is stopped, meaning unsubscribed to any events and unregistered at the Reconfiguration Engine.
- 1 = running. The component is running, meaning subscribed to certain events, registered at the Reconfiguration Engine and providing some functionality.
- 2 = quiescent. This provides a safe state for reconfiguration (e.g. suspending I/O), meaning unsubscribed to any events.

Components can currently only be deployed, undeployed and initially started by the action executor (via ant scripts on the command line or the Network Manager). Wire,

unwire, make quiescent, resume and stop however can be executed using a Reconfiguration (see the next section for further information on this). Starting a component for the 1 first time should be done using either the command line or the Network Manager as it requires the isolate to be initiated, once it is then made quiescent it can be restarted by sending a COMPONENTSTART event.

Furthermore, every component has a component ID. By default this is ComponentTypes.TEST_APPLICATION which translates to 0 but it is expected that every component implementation overrides the getComponentID() method and thus providing a more meaningful id.

Components can communicate with each other and the LooCI runtime via the Event Manager. The interfaces which components provide and require are defined by publishing and subscribing to certain event types.

4. A WSN-based River Monitoring System Using LooCI

There will be now an examination of an application of LooCI to a river monitoring system in the network of creeks that flows through São Carlos (São Paulo). In this application, WSN is deployed to measure the water depth and pollution levels of the creeks. In particular, we will investigate how new software and hardware technologies can be used to (1) improve the accuracy of flood warning forecasts, (2) support the designing of better environmental models and (3) respond effectively to flood-based events.

Details will first be given of the hardware prototype before we address the software aspects of our platform. It should be noted that our sensor network platform is called **REDE** which acronym of "**RE**de de sensores para **D**etecção de **E**nchentes". From now on, our platform will be referred to simply as REDE.

4.1. Hardware Components

4.1.1. Overview

Figure 1 shows a complete hardware prototype of our monitoring system. The prototype is based on SunSPOT; SunSPOT is particularly well suited to the rapid prototyping of sensor network applications as it runs standard Java ME (like many cell phones) and offers plenty of computational resources. The SunSPOT runs the LooCI tool which was described above.

After integrating the SunSPOTs with solar panels, batteries, depth and pollution sensors, the entire package was mounted in a weather-proof case and deployed on the creeks of São Carlos. The solar panels feed a large (12V, 7AH) reservoir battery, which in turn feeds the SunSPOT lithium-ion battery and this process maintains a continuous operation without the need for battery replacement. The deployed REDE mote is shown in Figure 1.

4.1.2. Use of Sensors

A vented-gauge hydrostatic pressure sensor is used to measure the water depth. Using this sensor, software components provide warnings when the water level approaches that

of the river-bank. in addition, conductivity-based pollution sensors are used by software computers to issue a warning whenever the pollution of the river is reaching an unacceptable level. We use an off-the-shelf conductivity sensor for industrial applications that provides a voltage output in the range of 1V to 5V that varies linearly with the level of conductivity. Finally, a three dimensional accelerometer is used by software components to detect human tampering and vandalism. The detection of vandalism and pollution are particularly beneficial in urban rivers such as the Tietê, which runs through densely populated areas.

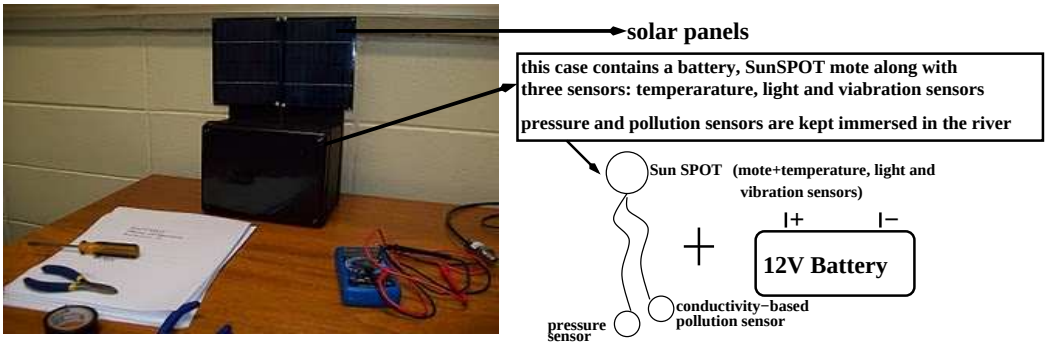

Figure 1. Our hardware prototype of the REDE node

Our platform has been built and an evaluation of REDE carried out at two creeks in the city of São Carlos. The deployment site is given in Figure 2, which shows two creeks and a tributary. Sensors were deployed before and after the confluence of the two creeks so that the relationship between the degree of pollution and the water level in each creek could be investigated. The base station (represented as BS) receives all the data from the deployed sensors, and acts as a data-logger and transmitting sensor readings back to the lab using 802.11b. Each REDE mote sensor transmits data to the base station at an interval of five minutes, which is the maximum sampling frequency recommended by our Environmental Science partners.

Figure 2. Map of the deployment site in São Carlos-SP. Deployment locations: BS = Base Station, C1 = Creek 1, C2 = Creek2, T1 = Tributary

4.2. The REDE Software Platform

The Loosely-coupled Component Infrastructure (LooCI) is a component model for Wireless Sensor Networks as mentioned earlier. LooCI allows for the creation of generic units of software functionality, known as "components" that communicate over an "event bus". By composing together generic components (e.g. a depth sensing component, a flow sensing component and a logging component), application developers can quickly develop rich application functionality (e.g. a flood monitoring application).

Our river monitoring application is implemented in four distinct modules: 1) Vibration module; 2) Depth sensor module; 3) Conductivity module; and finally the backend module that runs on the base station acting as a data-logger. Figure 3 shows our backend application running on the base station. Such an application logs data sent by pollution, depth and accelerometer sensors; it then updates that information on each textbox of the GUI. We constructed our application using NetBeans IDE with Java Sun 1.6 compiler toolkit.

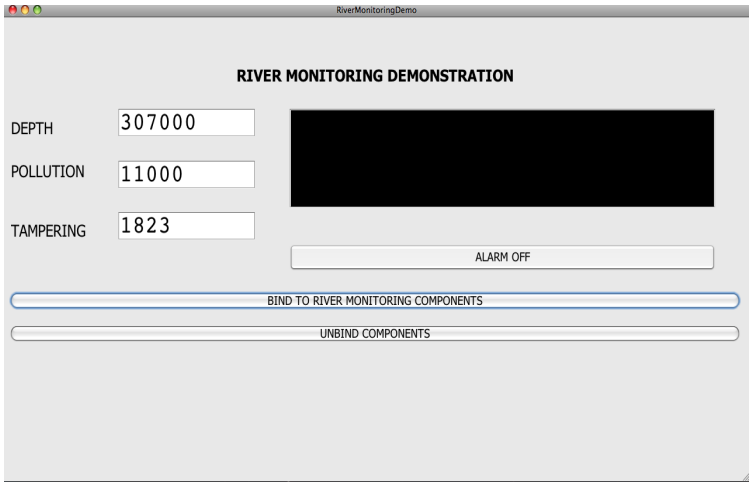

Figure 3. Our river monitoring application demo

5. Presentation Plans and Ongoing Work

This section outlines a tentative plan for the presentation of both the WSN component-based building technology and the application developed to validate the proposed tool.

The application itself that is based on LooCI is entirely implemented and has been demonstrated in seminars such as the one that took place at INPE (*Instituto Nacional de Pesquisas Espaciais*) recently. A project is now under construction to replicate this technology and deploy it along the Tietê river in São Paulo. The purpose of this is primarily to monitor potential pollution and flood hazards and provide people with a warning whenever they are at risk. With regard to the SBRC's tools session, our aim is to concentrate on three key aspects:

- *REDE Hardware Platform*. The complete hardware platform can be presented at the SBRC's tool session. This includes the SunSPOT microprocessor which is connected to the conductivity, depth and vibration sensors. The REDE kit also includes two solar panels that feeds the battery. Both SunSPOT and the battery are placed inside the weather-proof case as shown in Figure 1.

- *LooCI Component Model*. LooCI represents the underlying technology which enables us to construct lightweight runtime reconfigurable WSN applications. Our technology is based on component approach, which fosters modularity and more importantly, helps us to ensure that the application deploys components that are tailored to the target needs. The above-mentioned LooCI website includes a tutorial that can be exercised during the tool session.
- *River Monitoring Application-based on LooCI*. The last part concerns the presentation of the application of LooCI to constructing a real-world application scenario showing the benefits of our tool to the WSN domains. For this, we will show our river monitoring application running on a SunSPOT along with the conductivity, depth and vibration sensors.

6. Conclusions

This paper has provided an outline of LooCI – our tool for constructing runtime reconfigurable component-based applications to the domain of WSNs. We have also examined an application that adopted LooCI for building a WSN-based river monitoring system at São Carlos-SP. Unlike most of the existing models, our component model (i.e. LooCI) has an event-based binding type, which we argue reflects the nature of the WSN applications more closely. In WSNs, the connections between the nodes tend to be intermittent and thus, technologies should be able to handle unreliable environments. In SBRC's tools session, we plan to cover three aspects of our research: (i) the REDE hardware prototype, (ii) the LooCI component technology; (iii) a WSN-based river monitoring application written in LooCI. A general idea of our work can be found at `http://code.google.com/p/looci/` and `http://sp-river.blogspot.com/`.

Acknowledgements

Dr. Jó Ueyama and Dr. Daniel Hughes would like to thank FAPESP (Proc. 2009/01881-5 and 2008/05346-4) and CNPq (Proc. 474803/2009) for funding this research project. Wouter Horré is a PhD fellow of the Research Foundation - Flanders (FWO).

References

[1] P. Costa, G. Coulson, C. Mascolo, L. Mottola, G. Picco, and S. Zachariadis. A Reconfigurable Component-based Middleware for Networked Embedded Systems. *International Journal of Wireless Information Networks*, 14(2):149–162, June 2007.

[2] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A Generic Component Model for Building Systems Software. *ACM Transaction on Computer Systems*, 26(1), February 2008.

[3] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Programming Language Design and Implementation, ACM SIGPLAN*, 2003.

[4] D. Hughes, K. Thoelen, W. Horré, N. Matthys, S. Michiels, C. Huygens, and W. Joosen. LooCI: A Loosely-coupled Component Infrastructure for Networked Embedded Systems. In *Proceedings of the 7th International Conference on Advances in Mobile Computing & Multimedia (MoMM-09)*. ACM, December 2009.