

A middleware platform to support river monitoring using wireless sensor networks

Danny Hughes · Jo Ueyama · Eduardo Mendiondo ·
Nelson Matthys · Wouter Horré · Sam Michiels ·
Christophe Huygens · Wouter Joosen · Ka Lok Man ·
Sheng-Uei Guan

Received: 9 May 2010 / Accepted: 26 January 2011 / Published online: 15 February 2011
© The Brazilian Computer Society 2011

Abstract Flooding is a critical global problem, which is growing more severe due to the effects of climate change. This problem is particularly acute in the state of São Paulo, Brazil, where flooding during the rainy season incurs significant financial and human costs. Another critical problem associated with flooding is the high level of pollution present

in urban rivers. Efforts to address these problems focus upon three key research areas: river monitoring, modelling of river conditions and incident response. This paper introduces a rich next-generation middleware platform designed to support wireless sensor network based environmental monitoring along with a supporting hardware platform. This system has been deployed and evaluated in a real-world river monitoring scenario in the city of São Carlos, Brazil.

D. Hughes (✉) · K.L. Man · S.-U. Guan
Xi'an Jiaotong-Liverpool University, Suzhou, Jiangsu 215123,
China
e-mail: daniel.hughes@xjtlu.edu.cn

K.L. Man
e-mail: ka.man@xjtlu.edu.cn

S.-U. Guan
e-mail: steven.guan@xjtlu.edu.cn

D. Hughes · J. Ueyama · E. Mendiondo
University of São Paulo (USP), São Carlos 13566-585, Brazil

J. Ueyama
e-mail: joueyama@icmc.usp.br

E. Mendiondo
e-mail: emm@sc.usp.br

D. Hughes · N. Matthys · W. Horré · S. Michiels · C. Huygens ·
W. Joosen
Katholieke Universiteit Leuven, Leuven 3000, Belgium

N. Matthys
e-mail: nelson.matthys@cs.kuleuven.be

W. Horré
e-mail: wouter.horre@cs.kuleuven.be

S. Michiels
e-mail: sam.michiels@cs.kuleuven.be

C. Huygens
e-mail: christophe.huygens@cs.kuleuven.be

W. Joosen
e-mail: wouter.joosen@cs.kuleuven.be

Keywords Wireless sensor networks · Environmental monitoring · Middleware · Multiparadigm programming

1 Introduction

Flooding is a significant global problem that is likely to increase in severity due to the effects of climate change [1]. The problem of flooding is particularly acute in the state of São Paulo in Brazil, where during the rainy season from December to March, seasonal flooding incurs a significant financial and human cost. This problem is further complicated by the high levels of pollutants found in many of Brazil's urban waterways [2].

Wireless Sensor Networks (WSN) are a promising platform for supporting environmental monitoring. WSN are composed of tiny embedded computers known as “motes” that are equipped with low-power wireless networking technologies and simple sensors. These motes form ad-hoc, self-organising networks that are capable of sensing and reacting to the physical environment [3]. WSN promise a number of advantages for environmental monitoring compared to traditional telemetry systems. These include: (i) reduced cost, (ii) simplified deployment, (iii) high resolution sensing, and (iv) the ability to adapt to changing environmental conditions. These advantages are explored in more detail in Sect. 2.

The work presented in this paper improves upon current WSN approaches to environmental monitoring [4–6], by providing an integrated platform that is specifically tailored to the requirements of river monitoring in the state of São Paulo, Brazil. Our approach provides middleware support that respects the requirements and skills of the various actors, including: generic component developers, infrastructure owner/managers, application developers, and environmental scientists. To complement our work in the software domain, we also introduce a prototype hardware platform that demonstrates the feasibility of our approach to WSN-based environmental monitoring. In sum, these contributions provide a dependable and flexible solution that meets the requirements of river monitoring.

The remainder of this document is structured as follows: Sect. 2 provides background on WSN-based environmental monitoring. Section 3 provides background on the problem of river monitoring in the state of São Paulo. Section 4 describes our hardware sensing platform. Section 5 describes our software sensing platform. Section 6 evaluates the complete system. Section 7 discusses directions for future work. Finally, Sect. 8 provides conclusions.

2 WSN based environmental monitoring

Wireless sensor networks have evolved through two generations of technology. The *first generation* of WSN used custom hardware platforms and operating systems that were highly optimised for a specific application. This monolithic approach to early WSN, coupled with tight application integration made early WSN technologies difficult to reuse in other application domains. This first generation approach is exemplified by the Great Duck Island experiment that was performed by the University of California at Berkeley and the University of Maine who together deployed a sensor network in the burrows of Storm Petrels on Great Duck Island in order to monitor their nesting behaviour [4]. While the experiment at Great Duck Island suffered high rates of data loss and numerous technical difficulties, this experiment successfully showcased how low-cost sensor nodes, simple sensors and low power ad-hoc networking could be used to gather rich environmental data. The lessons learned from early experiments such as these would also be leveraged to create the second generation of generic sensor network technologies.

The *second generation* of WSN build upon the foundation of WSN Operating Systems (such as TinyOS [7], Contiki [8], and SOS [9]), the availability of standardised network protocols (such as 802.15.4 [10]) and the growing range of general-purpose sensor nodes (such as the MicaZ [11] and T-Mote [12]). Perhaps the most relevant of this second generation WSN is the Hydrowatch project

from the University of California at Berkeley which monitors several environmental factors using a large network of generic sensor nodes that are deployed in a river valley of the Sierra Nevada Mountains [6]. The Hydrowatch project demonstrates that WSN can operate efficiently at scale and that large numbers of off-the-shelf sensors combined with generic WSN systems software can be used to build a rich understanding of the environment.

In parallel to the deployment of second generation WSN applications, a number of research projects have investigated the use of more capable computing and networking technologies to implement environmental sensing infrastructure. For example, in the FloodNet project [13], the University of Southampton deployed a small network of powerful sensor nodes equipped with laptop-like CPUs and large amounts of memory that were networked using ad-hoc 802.11 b to monitor water levels on the River Crouch in southern England. While the use of laptop-like computing and network technologies provides flexibility and easy software development, these benefits come at a significant cost in terms of power consumption, which must be met by large solar arrays, connection to fixed power infrastructure, or frequent battery changes.

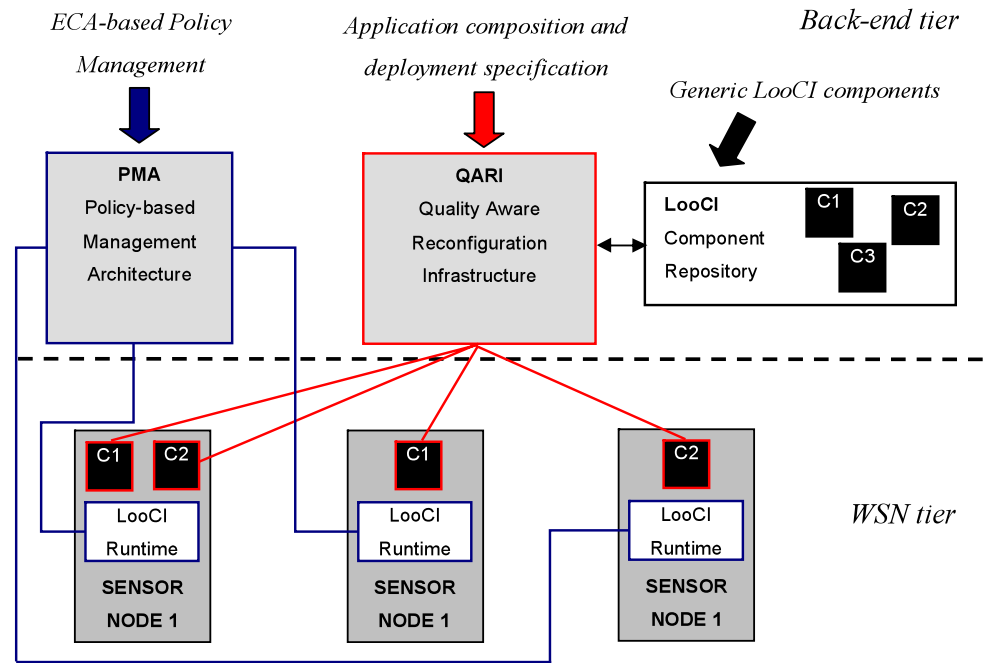
2.1 Emerging technologies

Most recently, a *third generation* of WSN have emerged that offer support for a rich range of programming paradigms. In our view, the most promising of these platforms include: *Java ME* [14, 15], *run-time reconfigurable component models* [16] and *policy-based management frameworks* [17].

Java ME The open-source SQUAWK virtual machine supports programming using the standard Java ME CLDC 1.0 specification [14] and has been successively ported to ARM-based sensor node platforms including the Sun SPOT. The Sun SPOT provides a 180 MHz ARM7 CPU, 512 KB of RAM and 4 MB of flash memory. The Sentilla Virtual Machine brings Java ME to even more constrained platforms, providing support for CLDC 1.0 on the Sentilla Perk sensor node, which offers an 8 MHz MSP430 MCU with 10 KB of RAM and 48 KB of flash [15].

Run-time reconfigurable component models The research community has been active in applying run-time reconfigurable component models to WSN problems. Run-time reconfigurable component models allow developers to dynamically modify a running application without having to restart the application. This is particularly advantageous in WSN scenarios where application development seldom ends neatly at deploy-time. Rather, the WSN developer often remains engaged in order to adapt system functionality to meet unexpected environmental conditions and to evolve system

Fig. 1 Development using DisSeNT



functionality to meet changing operational contexts. Key work in this field includes the OpenCOM component model [18], the Java version of which has been ported to powerful sensor node platforms such as the GumStix. OpenCOM was also the basis for the WSN-specific RUNES component model [19], which brought OpenCOM-like functionality to a T-Mote scale platform. Most recently, OpenCOM has been used as the basis for the run-time reconfigurable Lorien Operating System [20], which is suitable for T-Mote class devices.

Policy-based management of WSN Reconfigurable component models and Java ME are not generally a good fit with the skill-set of *system administrators*, who typically favour a more declarative, rule-based approach [17] such as that provided by a policy-based management architecture [21, 22]. For example, Finger [22] provides support for the policy-based management of TinyOS sensor nodes in a relatively small footprint. We believe that the need for policy-based management of WSN will only increase as these networks grow larger in scale, and are increasingly required to support multiple concurrent applications [17]. In a multi-application scenario where resources are shared between competing and cooperating applications, policies provide a high level mechanism for realising cross-cutting and system wide changes. The view of sensor networks as generic platforms that will support multiple applications throughout their lifetimes has growing support in the literature [23–26].

It should be noted that while these technologies are promising approaches, their usefulness to the varied stakeholders listed above will remain limited by the quality of

user-centric tools available. While we do not focus on this area of research in this paper, we consider it a priority for future research, as described in Sect. 7.

2.2 Our WSN approach

Our approach to WSN development builds upon our previous work on the DisSeNT middleware platform. DisSeNT provides LooCI, an efficient run-time reconfigurable component model [27], implementations of which are available for the Java-based Sun SPOT Platform [14] the Con-tiki based AVR Raven platform [8], and OSGi sensor nodes. DisSeNT offers three key programming abstractions:

1. The *Loosely-coupled Component Infrastructure (LooCI)* provides a run-time reconfigurable component model [27].
2. The *Policy-based Management Architecture (PMA)* provides an Event-Condition-Action (ECA) policy management framework [17].
3. The *Quality Aware Reconfiguration Infrastructure (QARI)* provides a high-level goal-oriented mechanism for specifying deployment requirements [28].

Figure 1 shows how LooCI, QARI, and PMA may be used together by developers to realise reliable and flexible WSN applications. With respect to our previous work in the area of WSN-based environmental monitoring [5], this paper presents a redesigned hardware and software platform that applies the latest advances in WSN and middleware technologies to meet the specific requirements of river monitoring in São Paulo state. We also provide a detailed experience report for this application scenario.

From a software engineering perspective, our work provides the first practical example of a multiparadigm programming approach for WSN that builds upon our previous work in policy-based management [17] and lightweight component models [27]. In terms of the LooCI component model, this scenario demonstrates how LooCI components can be used in combination with PMA policies to develop efficient applications that are manageable, maintainable and extensible. The features of our software system are described in Sect. 5.

3 River monitoring in São Paulo state

Flood hazards are the most frequent disaster in South American urban areas. Post-flood restoration plans transfer annual debts to population in a rate of US\$ 30 to US\$ 3,500 per capita, respectively, for small human settlements and metropolitan areas [29, 30]. In this way, South American countries are yearly impacted by a 2% to 5% decrease of gross domestic product caused by unprevented floods that become social disasters [29]. Accumulated losses of South American floods in the period 1995 to 2004 were US\$ 25 billion, including investment (INV) and Operation and Maintenance (O&M). Notwithstanding, alternative scenarios show that O&M costs could decay to US\$ ½ billion a year until year 2020 if early warning systems are included in river committee plans, with increased monitoring in today's ungauged basins. This nonprofited "flood market" with prevention and monitoring programs in South America is around US\$ 80 billion a year, and the Brazilian flood market is approximately US\$ 40–50 billion a year [31]. Over the last 100 years, millions of Brazilian people have been affected as their housing, sources of income and communities were damaged or destroyed by extreme floods [32, 33]. In January 2004 and in January 2010, Brazilian floods raised losses to approximately US\$ 10 billion and US\$ 30 billion, respectively. During January 2010 floods, the São Paulo Metropolitan area suffered continuous flooding, encompassing ca. 700 mm a month. Flood waters carry out not only high volumes of runoff, but transfer pollution and widespread contamination sources from uplands to lowlands of the same watershed [34]. At an urban drainage system located in headwaters of São Carlos city, pollution levels of total coliforms can raise to 20 million colony-formed units and biological oxygen demand (BOD) to levels higher than 100 mg L^{-1} , raising serious concerns over contamination and eutrophication [35]. Common runoff from urban creek with 1-Km^2 drainage area can attain strong pollution rates of total coliforms loads between 10^9 to 10^{12} colony-formed units per second and with BOD loads up to $1/2 \text{ Kg per second}$ [36].

Wireless sensor networks that are capable of sensing and reacting to water and pollutant levels hold the potential of

significantly reducing the damage, health-risks and financial impact of flood events. The specific requirements of river monitoring in the state of São Paulo are explored in Sect. 3.1.

3.1 Requirements of river monitoring in São Paulo

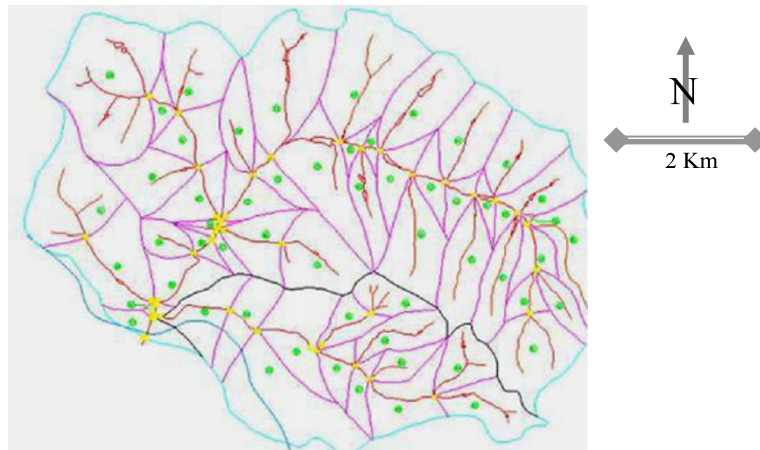
To monitor river conditions of a quantitative and qualitative nature, specific requirements must be met in terms of (i) Nested Catchment Experiments (NCE), (ii) ecohydrological features of drainage systems, and (iii) early warning systems to reduce vulnerability at flood-prone areas.

The concept of *Nested Catchment Experiments* (NCE) [37, 38] necessitates the deployment of monitoring stations according to the occurrence of the hydrologic cycle across the river basin, says from headwaters to lowlands. The NCE concept is the most popular approach among hydrologists in order to compare natural behaviour and thresholds across river catchment, especially to make predictions at intermediate scales of ungauged or poorly gauged basins [36]. Another advantage of NCE is the possibility to flexibly change stations, along the drainage systems, in order to capture the complexity of processes in critical points of urban areas. To support this, sensor networks for environmental monitoring must therefore offer mechanisms to simplify the management of large-scale gauging networks and minimise maintenance overhead. Our approach provides support for large-scale management through network-wide reconfiguration and policy based administration. The sustainable power supply of our sensor node platform also reduces maintenance overhead due to battery replacement as does the robust nature of our hardware platform.

The *ecohydrology* features of drainage systems are often poorly characterised in areas undergoing urbanisation, for example, in terms of Biological Oxygen Demand (BOD), Chemical Oxygen Demand (COD), Total Suspended Solids (TSS), and electric conductivity. Sensor networks for river monitoring must therefore offer support for a range of relevant sensors, and mechanisms to facilitate the integration of new sensor types. Our approach provides support for diverse sensors through the use of an extensible hardware platform and a semantically-rich event model, which allows for the easy integration of diverse sensor data into semantically aware applications. Furthermore, the ability of our platform to offer flexible duty-cycling without battery replacement allows for the use of sensors that demand high sampling rates.

Accurately assessing *vulnerability criteria* is critical for early warning systems, either using participatory-based or emergency-based layouts. For instance, the combined features of direct estimates of water levels and indirect average velocity vectors at a given location can provide a vulnerability index for movable objects like cars or even individuals living in flood-prone areas. The strength of this approach is that it could merge wireless monitoring systems

Fig. 2 (Color online) Map of drainage basins of São Carlos, SP, Brazil



into urban policies and strategies to reduce both risks and thus costs. Furthermore, local flood signboards can inform citizens of the best transport options when floods are imminent or in progress. To support this, sensor networks for river monitoring must therefore offer support for integration with other local devices and also back-end systems. We provide support for this integration using a rich semantic representation of sensor events and a system-wide event bus, which can be used to support communication between any system elements. Figure 2 shows 54 river sub-basins located in São Carlos city, all linked to the drainage system and natural topology of river networks, with headwaters and creeks. Runoff tributaries from headwater areas (●) are separated by boundaries that are linked to drainage systems at confluence points on valleys (yellow points), topologically forming the entire river basin network.

We have chosen to evaluate our approach to supporting flood monitoring and warning in São Paulo state using a small scale deployment in São Carlos. This deployment is highlighted in Fig. 3 and described in detail in Sect. 3.2 below.

3.2 Deployment environment

For our initial prototype deployment, we have chosen to monitor the intersection of two creeks, which are shown in Fig. 3. The maximum separation between any two sensor nodes in this deployment was 50 m and all sensor nodes were able to communicate directly with each other. The base-station {BS} was permanently powered and connected to a fixed Internet link, while all other sensor nodes {T, C1, and C2} were powered only using solar arrays and communicated using an 802.15.4 mesh network. All motes with the exception of {BS} were mounted on trees near the river at a height of 3 m. While this deployment is of small scale, it provides a realistic environment in which we can evaluate the functionality of the proposed hardware and software platform.



Deployment Locations: BS = Base Station, C1 = Creek 1, C2 = Creek 2, T1 = Tributary

Fig. 3 Prototype deployment location

4 Hardware sensing platform

While the core contribution of this paper is in the software field, in order to demonstrate the feasibility of our approach it was necessary to create a custom hardware platform. This platform is built around the Sun SPOT sensor node [14], which was extended with solar power facilities, expanded battery capacity, and additional sensors. We refer to this integrated river monitoring package as REde de sensores sem fio para Detectar Enchentes (REDE). Section 4.1 provides an overview of the specifications of the REDE platform and Sect. 4.2 compares this to the GridStix platform [5] that we developed and evaluated during our previous river monitoring work.

4.1 REDE hardware specifications

Based upon the requirements of our environmental science partners, we have opted for a powerful sensor node platform that can meet the local processing needs of advanced sensors

and on-site modelling. The core of the REDE platform is the Sun SPOT sensor node [14]. In terms of computational specifications, the Sun SPOT offers a 32 bit ARM 7 CPU running at 180 MHz with 512 KB of RAM and 4 MB of Flash Memory. With respect to networking, the SPOT offers a standard IEEE 802.15.4 radio [10]. The SPOT offers also offers the following sensors: temperature, light, and a three-dimensional accelerometer. In terms of sensor expandability, the SPOT provides six 10 bit analogue input channels, two digital input channels and two digital output channels. Our additions to this platform are summarised below.

Solar power A key requirement of our environmental science partners was that REDE sensor node should be able to operate continuously without battery replacement. This necessitates scavenging of power from the physical environment. To meet this requirement, we considered using micro wind generators, micro water turbines, and solar panels. After some investigation, we settled on solar power, due to the plentiful solar energy available during Brazil's summer rainy season and the high comparative cost of micro turbines and micro wind generators. Specifically, we integrated a 15 CM \times 24 CM solar array constructed from a pair of monocrystalline panels This array produced a maximum of 150 mA at 15 V and was used to trickle-charge a large backup battery (see below). We opted for monocrystalline panels as these offer high efficiency at a relatively low price point [6]. The performance of our solar power harvesting hardware is analysed in Sect. 6.1.1.

Secondary battery In addition to the internal lithium ion battery available on the Sun SPOT, we integrated a 7 AH, 12 V sealed gel lead-acid battery. This type of battery has three key advantages in WSN scenarios: (i) low price point, (ii) high charge density and (iii) robustness. Despite these significant advantages, sealed gel batteries are not commonly used in WSN scenarios due to their size [6]. In this case, however, our environmental science partners place a significantly higher value on a low price point and robustness than the size of an individual sensor node. As will be shown in Sect. 6.1.3, this battery has ample capacity to serve as a reliable reservoir for the energy that is produced by our solar array.

Sensors In order to monitor water depth, we used a vented gauge pressure sensor (SSI model P51-15). We also integrated sensors to test water quality including a custom-built conductivity sensor and a methane sensor (HANWEI model MQ-4). These sensors were integrated over the analogue input channels of the REDE sensor node. In addition, the internal accelerometer of the REDE sensor node was used to implement tamper and theft detection.

4.2 Comparison with the GridStix sensing platform

This section compares the REDE sensor node with our previous hardware platform, GridStix which was deployed in two flood monitoring and warning scenarios in North–West England and North Wales between 2005 and 2009 [5]. In terms of hardware, there are three key differences between the REDE platform and the GridStix platform:

Computational platform The GridStix used a powerful 400 MHz XScale CPU, while REDE uses a more modest 180 MHz ARM 7 CPU. As the ARM 7 CPU consumes less power, a smaller solar array is needed, resulting in a lower price-point for REDE sensor nodes and more flexible duty-cycling.

Networking The REDE platform uses a more energy efficient, WSN-specific IEEE 802.15.4 radio rather than 802.11 b/g as used in GridStix v1 or the mixed approach used in GridStix v3 [39]. This further reduces power demand, and thus the price-point of REDE sensor nodes.

Sensing While GridStix provide support for only pressure and flow sensors, REDE provides support for a number of new sensors including depth, conductivity, methane, and tamper-detection, as described in Sect. 4.1.

The REDE platform is able to meet the demands of our river monitoring application using a smaller, more power efficient platform in large part because of our more efficient software architecture, which is described in Sect. 5.

5 Software sensing platform

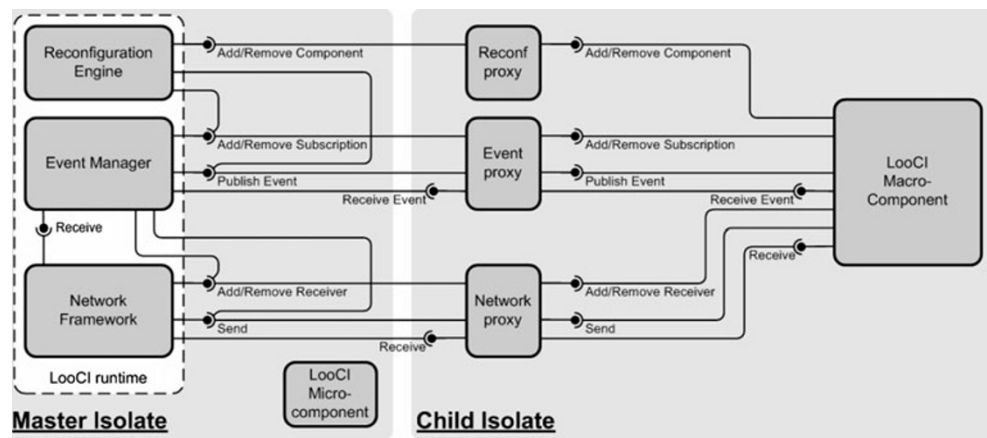
Our approach to designing WSN software is based upon the DisSeNT middleware framework, as described in Sect. 2.2. DisSeNT offers three primary tools that will be exploited in the context of this project (i) a reflective run-time reconfigurable component model known as LooCI [27], (ii) a high-level policy management architecture known as PMA [17] and (iii) a quality aware deployment mechanism known as QARI [28]. This section discusses how these features were used in concert to implement, deploy, and manage the river monitoring application.

5.1 Multiparadigm WSN programming

Typical WSN scenarios involve multiple actors such as: the infrastructure owner/manager, generic component developers, application developers, and application users. In the context of the presented river monitoring scenario, the actors are as follows:

Developers of generic components Such as sensing, logging, and networking. This role was filled by Software Engineers at Xi'an Jiaotong-Liverpool University using the LooCI component model [27].

Fig. 4 The Loosely Coupled Component Infrastructure (LooCI)



Managers of WSN infrastructure Specialise in the provision and management of hardware infrastructure. This role is filled by Computer Scientists at the University of São Paulo who use PMA [17] to manage deployment-related concerns, such as software access rights and power management.

Application developers Use the high-level composition specification language included with QARI to compose generic components into a coherent application. This role was filled by Computer Scientists at the University of São Paulo.

Environmental scientists Interact with the system in two key ways. At deployment time, they provide a high level environment specification that is used to achieve a quality-aware software deployment. At run-time, Environmental Scientists may tailor system behaviour in order to adapt to changing environmental conditions and application requirements. This role was filled by Environmental Scientists at the University of São Paulo.

More details on how these development tools are used to realise application functionality are provided in Sects. 5.2 to 5.5.

5.2 Component based development with LooCI

The Loosely-coupled Component Infrastructure (LooCI) is a run-time reconfigurable component and binding model that is designed for networked embedded systems such as WSN. Along with a small memory footprint and good performance, the key benefits of LooCI are that it promotes an event-based interaction paradigm and a loose coupling between cooperating components. We believe that these features are a good fit with WSN, which are inherently asynchronous and unreliable. LooCI provides these features using an event-bus communication abstraction supported by a connectionless networking model [27]. In the *LooCI component model* an interface is modelled by the publication of a specific event type on the event bus, while a receptacle is

created by the subscription of a component to a specific type of event. The LooCI component model is platform and language independent, with ports available for the Java-based Sun SPOT platform, the Contiki-based AVR Raven platform as well as GumStix and standard PCs [40]. LooCI supports *runtime reconfiguration* including: (i) dynamic component deployment, (ii) distributed mechanisms to start/stop a component and to place components into quiescent state and (iii) support for the dynamic modification of bindings between components.

The *LooCI event bus* follows an entirely decentralised publish-subscribe architecture, wherein an “Event Manager” on each node acts as a broker—disseminating incoming events from the network to local components with a matching receptacle and disseminating outgoing events to the appropriate location. Addressing of entities on the event bus is of flexible granularity. At the finest level of granularity, a component interface may be bound to a specific receptacle available on a specific component that is hosted on an individual sensor node, while at the coarsest level of granularity; a component interface may be bound to all matching receptacles available within an entire network (for a detailed description of LooCI components, bindings, interfaces, and receptacles, we refer the reader to [27]). Section 5.2.1 shows the event types used in the river monitoring application and their associated ontology

An architectural overview of the LooCI runtime is shown in Fig. 4. The Sun SPOT LooCI port builds on the notion of isolates, which provide process-like units of encapsulation [41]. From Fig. 4, it can be seen that the LooCI run-time is hosted in the *master* isolate, which executes on start-up. LooCI offers two types of components. *Macro-components* each run in a separate isolate. They may be multithreaded and can include utility classes and data files. *Microcomponents* are delivered in a child isolate, but are then transferred to the master isolate, where they execute alongside the LooCI runtime. Microcomponents must be single threaded and self-contained. Microcomponents offer

better performance than macrocomponents [27]. Aside from these distinctions, macrocomponents and microcomponents are functionally identical. In this project, it was only necessary to use microcomponents.

In terms of interaction paradigms, LooCI is designed to support applications with periodic or event-based data requirements, which we believe includes the majority of WSN applications. Query-based applications may be created using LooCI, however, no implicit support is provided for this.

5.2.1 Component-based composition

LooCI events are defined based upon a global, hierarchical name-space represented by a spanning tree. The root of the tree is the base EVENT type and each successive layer of child nodes describes events with increased specificity. The structure of this tree is encoded using a unique identifier which can be used to support efficient subsumption testing, i.e., the testing of whether one event is a sub-type of another. This scheme is based upon the work reported on in [42]. In brief, each event in the tree is assigned two values, (n) and (p) which together can be used to efficiently compute a unique identifier (b) that is used to encode subsumption testing functionality into each event:

n : represents the sequence number of the event as it was added to the tree. The first event is assigned the value 1, the second event 2, and so on and so forth.

p : is the n th prime. This is not computed dynamically, but instead is assigned based upon a well-known list of the first 10,000 primes [25].

b : is the Globally Unique Identifier (GUID) for this event type and is calculated based upon the product of the assigned p -value and the b -value of the parent node. The root node is a special-case and is assigned the b -value 1.

The portion of the event ontology that relates to sensor events is shown in Fig. 5 along with the unique identifiers that are assigned to each event (b -values). For space reasons, the complete ontology is not reproduced here.

As reported in [42], due to the properties of primes, it is easy to test whether one event (X) is a subtype of another event (Y). All that is required is to calculate the modulus of the b -value for X by the b -value for Y (i.e., $X.b \text{ Mod } Y.b$). If the result of this operation is 0, then X is a subtype of Y , otherwise X is not a subtype of Y . This event taxonomy is exploited to allow for the creation of a generic *Environmental Hazard Alert* component that subscribes to all events that are sub-types of SENSOR using a single receptacle.

The river monitoring composition consists of five components, which are bound as shown in Fig. 6. Lines ending in balls denote *interfaces*, while lines ending in cups denote *receptacles*. The number above each interface/receptacle indicates its type as defined by the n -number assigned in the application ontology and illustrated in Fig. 5.

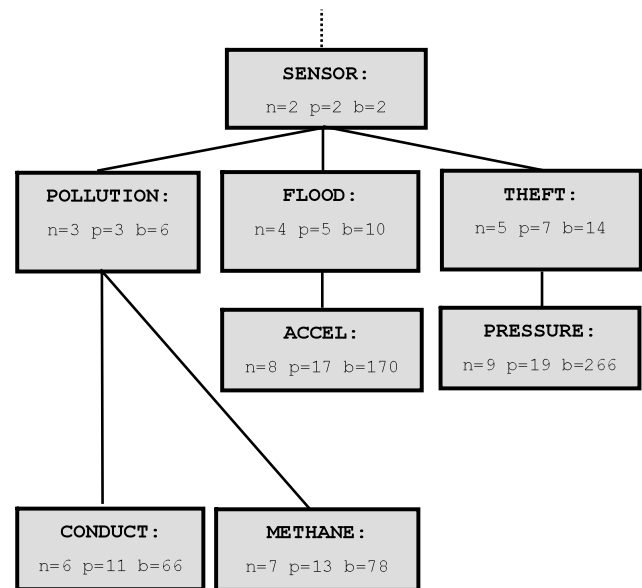


Fig. 5 Event ontology

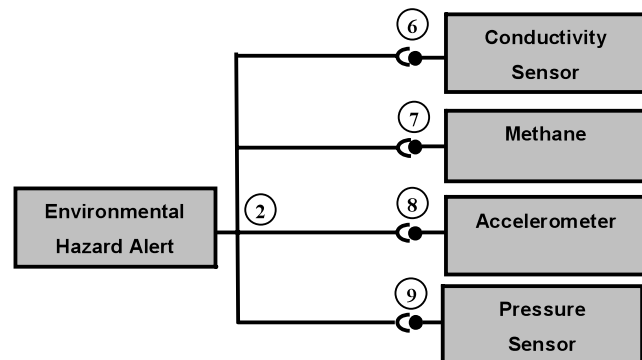


Fig. 6 Pure component-based application composition

As can be seen from Fig. 6, in the pure component-based composition, the Environmental Hazard Alert component subscribes to sensor data from all other components and will receive sensor data at a rate controlled by the individual component parameterisations.

5.3 Policy-based refinement with PMA

The Policy Management Architecture (PMA) [17] integrates with the LooCI event bus [27] in order to provide support for tailoring and management of system functionality through the use of high level policies. As all interactions between LooCI components occur over the event bus, it is possible to tailor any aspect of system functionality by modifying the manner in which events are allowed to propagate. To facilitate this type of application management, the LooCI runtime is extended with a compact policy engine, which executes a simple event-condition-action policy specification language.

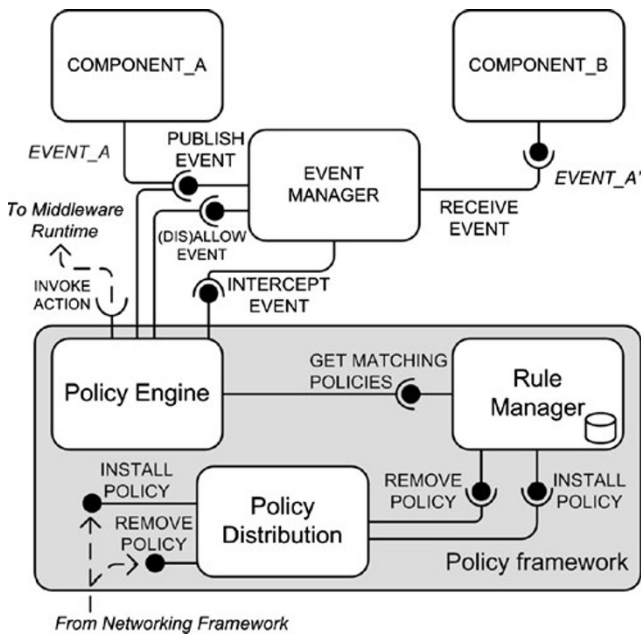


Fig. 7 The policy-based management architecture

The PMA *Policy Engine* is integrated with the per-node LooCI Event Manager (as shown in Fig. 7) and intercepts all event flows. Where an event is found to match the target listed in a Policy Specification, the policy engine may apply a number of operations. Passing events may be: *blocked*, *redirected* to a new destination, *transformed* into a new event type or otherwise allowed to pass without modification.

The *Policy Specification Language* used in PMA is high-level and based upon Event-Condition-Action (ECA) rules. This is a good fit with the typical skill-set of a system administrator and is supported by a simple back-end tool that allows policies to be applied at various levels of granularity [17]. Consider the following examples:

At the granularity of an *entire network*, an administrator may block all reconfiguration events in order to place the system into a “stable” operational state. At the granularity of a sensor node, an administrator may deny all outgoing events that are destined for sensor nodes not owned by the same party in order to provide security for sensitive infrastructure. At the granularity of *component interfaces*, an administrator may allow a 3rd party to access the data produced by a sensing component, but not allow those same users to place the component into a nonoperational state. This would be achieved by denying all incoming reconfiguration events that did not originate from the system administrator. Section 5.3.1 describes how PMA was used to tailor and manage the component functionality described in Sect. 5.2.

5.3.1 Policy augmented composition

The river monitoring application composition includes four policies, the functionality of which is outlined below:

ConductivityDetectionPolicy This policy effectively acts as a filter, allowing CONDUCTIVITY (6) events to pass to the *EnvironmentalHazardAlert* component only where the reading contained in the EVENT exceeds a specified threshold (and thus indicates that a conductive pollutant has entered the river). This threshold value is configured at the level of individual sensor nodes, based upon previously observed readings.

MethaneDetectionPolicy This policy effectively acts as a filter, allowing METHANE (7) events to pass to the *EnvironmentalHazardAlertComponent* only where the reading contained in the EVENT exceeds a specified threshold (and thus indicates that an organic pollution event has occurred). This threshold value is also configured at the level of individual sensor nodes, based upon previous readings.

TamperDetectionPolicy This policy filters ACCEL (8) readings, only allowing them to pass to the *EnvironmentalHazardAlertComponent* where they exceed a threshold that indicates theft or vandalism. This threshold is initially configured globally for every sensor node, but may be tailored at run-time to account for natural movement of the sensor nodes (for example, due to trees swaying in the wind).

FloodDetectionPolicy This policy filters PRESSURE (9) readings, only allowing them to pass to the *EnvironmentalHazardAlertComponent* where they exceed a threshold that indicates flooding. This threshold value is configured at the level of individual sensor nodes, based upon the height of the river bank at that deployment location; however, this value may be tweaked to account for bank erosion or reinforcement.

The complete application composition including policies and components is shown in Fig. 8. As can be seen from Fig. 8, components are used to provide generic functionality, including the encapsulation of physical resources like sensors and generic tasks like power management and logging. In contrast, policies are used to tailor this functionality, for example, by setting a specific level at which sensor readings should generate an alert to the application back-end. An example filtering policy is provided in Listing 1.

Data interpretation occurs in the resource-rich back-end, where it is performed by the Environmental Hazard Alert component, which applies computational models to determine the current risk due to pollution, flood, and theft. In the case of flood and theft, these models need only a single type of sensor data (pressure and accelerometer, respectively), while in the case of pollution, both conductivity and methane data are required in order to better characterise the

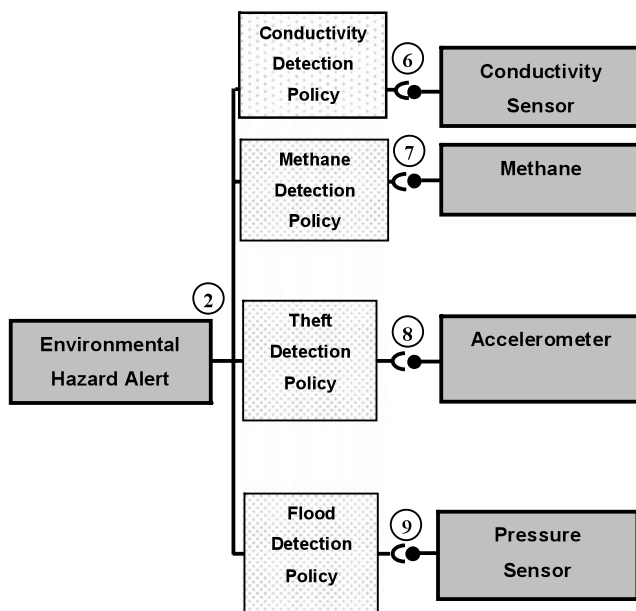


Fig. 8 Mixed policy and component-based composition

```

policy "flood detection" {
  on PRESSURE_SENSOR_READING_EVENT as p;
  if( p.value >= 500 )
  then(
    publish SENSOR_READING_EVENT(p.eventID,
      p.src_comp, p.value);
  )
}
  
```

Listing 1 The flood detection policy

pollution event that has occurred. We consider these models outside of the scope of this paper and would refer the interested reader to our previous work in this area [43].

5.4 Software deployment and maintenance

Software deployment is achieved using the tools provided by the Quality Aware Reconfiguration Infrastructure (QARI) [28]. Following application composition, quality aware deployment specification is achieved using two high-level domain-specific languages. First, a *target network* specification is provided using the high level network description language of QARI. This language allows the application developer to specify regions, which may be nested and to map nodes onto these regions. Secondly, a *quality aware deployment specification* is provided that describes how software should be deployed within the regions created in the first stage. This additional layer of abstraction simplifies the deployment process and allows for the automatic repair of compositions at run-time.

This information is then used by the per-network QARI run-time, which executes on the network gateway and attempts to honor and enforce the intentions of the deploying actor. After deployment, the QARI runtime will poll the network using the reflective functionality of LooCI [27] and compare observed component availability against the required quality aware deployment specification. Where a region is found not to meet the requirements defined in the associated *quality specification*, repair action will be undertaken. For example, if the developer specified that exactly one node in the region “RIVER_A” should report the water level, appropriate components, and policies would be deployed by QARI to that node. If that node were then to fail, QARI would detect this mismatch and deploy the appropriate components to another sensor node in “RIVER_A.” If it is not possible to repair the deployment to meet the provided quality specification, then QARI will disseminate an exception to the actor who is responsible for the deployment.

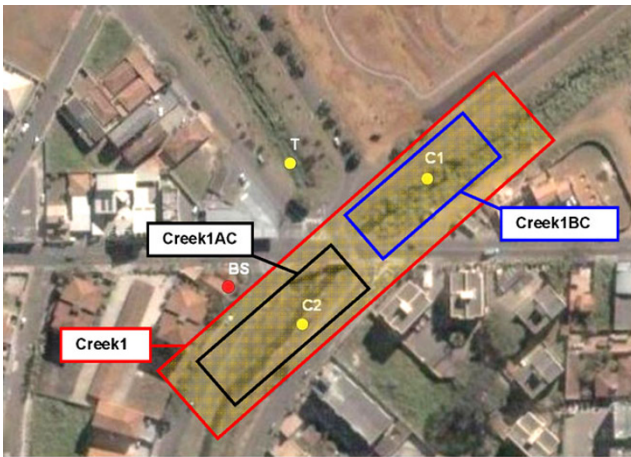
5.4.1 Target network specification

The target deployment location is described in detail in Sect. 3.2. In brief, our test deployment consists of four sensor nodes deployed over an area of approximately 100 square meters, covering the confluence of two creeks in the city of São Carlos. As described in Sect. 3.2, we deploy a sensor node on each creek upstream of the confluence, and one sensor node downstream of the confluence in order to infer the relationship between water levels and pollution levels in the two creeks. To support easy software deployment, maintenance, and management, we divide the creek into a number of regions. Figure 9 shows the deployment region graphically, while Listing 2 provides a subset of the QARI target network specification that was used to realise zoning for this application scenario. Due to space constraints only a subset of regions are described.

5.4.2 Quality aware deployment specification

The River Monitoring application is composed of a combination of components and policies that are connected together as described in Sect. 5.3.1. This composition is realised using the quality-aware deployment specification language provided by QARI. The deployment specification states that (in the interests of brevity, a full listing of the deployment specification has been omitted):

- ConductivitySensor components should be deployed to one node in each sub-region of *Creek1*.
- A EnvironmentalHazardAlert component should be deployed to the region *BaseStation*.
- All ConductivitySensor components should be bound to an EnvironmentalHazardAlert component. This will cause ConductivityReadings to be transmitted to the alert system.



Deployment Locations: BS = Base Station, C1 = Creek 1, C2 = Creek 2, T1 = Tributary

Fig. 9 Deployment topology regions

```

Group Creek1Nodes {
  desc = "All nodes at Creek 1";
  nodes = [$C1,$C2]; }
Group Creek1BeforeConfluenceNodes {
  desc = "Nodes on Creek 1 before the
  confluence";
  nodes = [$C1]; }
Group Creek1AfterConfluenceNodes {
  desc = "Nodes on Creek 1 after the
  confluence";
  nodes = [$C2]; }
Region Creek1 {
  // This region includes two sub-regions
  desc = "Creek 1 (both before and after
  the confluence)";
  Region Creek1BC { // Before Confluence
    desc = "Creek 1 before the confluence";
    coveredBy = $Creek1BeforeConfluenceNodes;}
  Region Creek1AC { // After Confluence
    desc = "Creek 1 after the confluence";
    coveredBy = $Creek2AfterConfluenceNodes;}
}
    
```

Listing 2 The target network specification for this application

- A `ConductivityDetection` policy should be bound to all `ConductivitySensor` components in order to allow the customisation of the conductivity levels that generate a hazard alert.

As described in Sect. 5.5, following the deployment and binding phase, QARI will actively attempt to maintain compositions at run-time. For example, if the `ConductivitySensor` was deployed to a single node in the region `Creek1` and this node was to fail, QARI would attempt to deploy another `ConductivitySensor` and bind this to `EnvironmentalHazardAlert` component instead.

5.5 Maintenance and management using DisSeNT

Following the deployment stage, system maintenance and management is accomplished by using QARI to deploy new or updated policies and components. Policy updates are used to implement fine-grained or cross-cutting changes, while component updates are used to implement long-lived evolution of system functionality. In a well-crafted DisSeNT composition, policies should be used to represent all of those concerns that would typically be tailored at runtime by users or infrastructure managers. Unanticipated concerns may be addressed by the injection of a new policy at runtime. A number of illustrative examples follow:

Example 1 Tailoring the issuance of flood alerts At deployment time, the water level that generates a flood alert upstream of the confluence on Creek 1 was estimated based upon some fraction of the height of the bank; however, empirical data from the deployment location showed that this initial setting generated too many false flood warnings. To address this, an update of the flood alert policy was deployed using QARI to all nodes in the region `Creek 1 BC`.

Example 2 Modifying component functionality The performance of the depth sensor was subject to short-lived inaccuracies due to waves that occur in Creek 1 (these do not occur in Creek 2). To address this, an update of the `PressureSensor` component that uses time averaging to negate the effects of waves was deployed to all nodes in the region `Creek 1` using QARI.

Example 3 Injection of logging functionality at run-time To allow for roll-back and data recovery in the event of network failure, it was decided that logging functionality should be added to the composition. To accomplish this, a generic logging component, which logs events of type `SENSOR` was first deployed to a subset of sensor nodes. A new policy was then deployed to all sensor nodes, which forward copies of all `SENSOR` events to the logging component.

It is important to note that without the policy based management architecture offered by DisSeNT, these modifications would require the ongoing redevelopment of components.

6 Evaluation

We have quantitatively assessed the performance of REDE sensor nodes in terms of *power characteristics*, *sensor support*, *application footprint*, and *developer overhead*. The results of this evaluation are reported in Sects. 6.1 to 6.4, respectively. All of the results reported in this section were



Fig. 10 A REDE sensor node being tested at the deployment site

gathered in the context of the real-world deployment environment described in Sect. 3.2. A complete REDE sensor node is shown being tested at the deployment site in Fig. 10.

6.1 Power characteristics

This section firstly evaluates the *power production* capabilities of the solar array provided with the REDE sensor node. We then analyze the *power consumption* of the REDE platform and use this data to derive the *minimum duty cycle requirements* that allow for continual operation of the REDE sensor nodes without battery replacement. This evaluation is necessary to demonstrate the feasibility of our approach for this application scenario.

While it is clearly important to also evaluate how quickly and reliably the REDE system can generate flood and pollution warnings, performing such an evaluation is difficult due to: (i) the inherent tension between providing timely warnings and reliably warnings as explored in [43], (ii) the need for a large-scale deployment to gather sufficient experimental data to support this evaluation and (iii) the need to capture environmental data over a long period in order to capture sufficient flood events. Due to these complexities, we consider this evaluation to be a key area for future work, as explored in Sect. 7.

6.1.1 Solar panel power production

To provide a realistic assessment of the power scavenging capabilities of our solar array (described in Sect. 4.1), we evaluated the array at the deployment site. The physical constraints of the deployment site did not allow for optimal placement of the solar array; necessitating that it be mounted facing south easterly and with an inclination perpendicular to the horizontal. For some parts of the day, the panel was also partially shaded by vegetation.

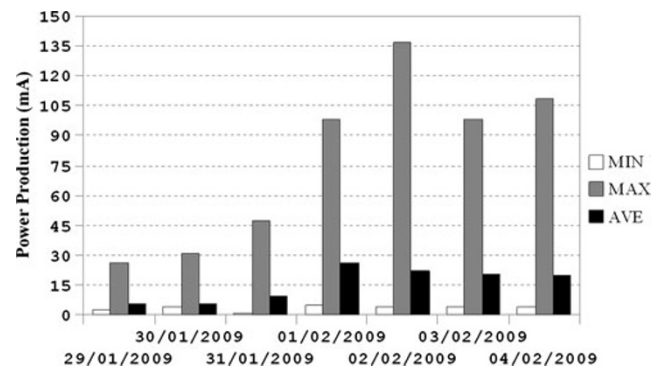


Fig. 11 Solar panel power production

Figure 11 shows the minimum and maximum power production recorded during daylight hours. The *average* value includes the night time, when no solar power is available. The *minimum* value refers to the lowest solar power level observed during the day time. Weather conditions varied throughout the evaluation period. From the 29th to the 31st of January, stormy weather with heavy rain and dark cloud cover prevailed. From the 1st to the 4th of February, weather conditions were sunny with light clouds. We are confident that the combination of harsh weather conditions and sub-optimal placement of solar panels make our results representative of typical solar conditions that might be expected during the São Paulo rainy season.

The maximum power production recorded during our experiments was 136 mA on the 2nd of February, while the average power produced during the experiment was 15 mA, including periods of darkness.

6.1.2 REDE power consumption

We recorded the power consumption of the REDE sensor nodes in the lab during three key phases of operation:

Sensing phase In this phase the sensor interface board is operational so that data may be read from the attached environmental sensors.

Transmission phase In this phase the sensor node is actively transmitting data to a remote entity using the 802.15.4 radio. During this phase, the sensor board is disabled.

Sleep mode In which the sensor node hibernates in order to conserve power, for example during the “off” period of duty cycles.

During all experiments, the REDE sensor nodes ran the DisSeNT middleware and the default Sun SQUAWK JVM, “Red” version. Power consumption was recorded at an interval of 250 ms and averaged over a period of one minute. The average power consumed by the REDE sensor nodes in *sleep mode* is 35 μ A, while the power consumption during the *data transmission* phase averages 75 mA. During the *sensing phase*, the sensor node consumes an average of

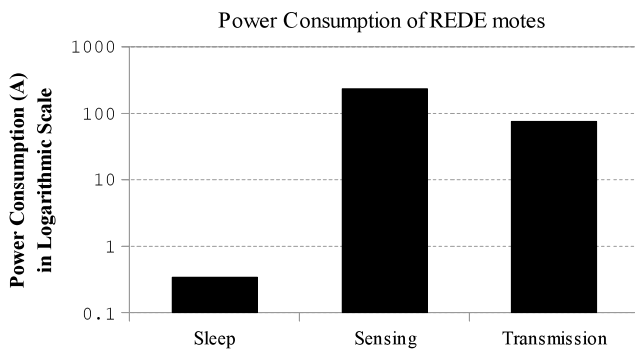


Fig. 12 REDE power consumption

230 mA. The power profile of REDE sensor nodes is shown in Fig. 12.

It should be noted that, for REDE sensor nodes, the sensing phase of activity consumes significantly more power than the transmission phase, which is unusual for sensor node platforms [44]. This discrepancy arises because the REDE platform incorporates complex sensors with significant power demands, such as the methane sensor which uses a small heating element as part of the gas detection process. In contrast, most sensor node platforms reported on in the literature use only simple sensors with low power demands [12, 15, 41].

6.1.3 Minimum duty cycle requirements

In order to ensure continued sensor node operation in the face of limited power availability, sensor nodes must be duty-cycled (i.e., periodically placed into sleep mode). The literature provides significant work in the area of duty cycling and a number of approaches exist to ensure power conservation in the face of changing environmental conditions and operational contexts such as [45] or [46]. Whatever duty-cycling approach is selected, the parameterisation of this approach will be limited by the power-consumption and generation capability of the sensor node platform. This section discusses the minimum duty cycling requirements that REDE sensor nodes impose upon applications. Using the average power production of our solar panels (15 mA) as the *upper bound* to our power budget, we analysed the length of time for which each sensor node must sleep, following specific periods of transmission and sensing.

Due to space considerations, Table 1 only shows short sensing and transmission periods, however, we believe that this time-scale is appropriate for environmental monitoring applications. As can be seen from Table 1, a safe minimum duty-cycle period may be easily deduced based upon the desired length of the sensing and transmission phases. For example, with a transmission phase of 5 seconds and a sensing period of 5 seconds, a REDE sensor node must then sleep for 92 seconds before resuming operation.

6.2 Component-based sensor components

The pressure, conductivity, and methane sensors used in this scenario provide a voltage output, which may be read from the ADC of the REDE sensor node using a simple call to the underlying SQUAWK API that can be linearly converted into an associated data point. Because of the simplicity of these components, their performance is not analyzed here. In contrast, the tamper detection component uses more complex three dimensional accelerometer data and required specific experimentation. We have therefore chosen to use the *TamperDetection* component and the associated policy as an illustrative case-study that demonstrates component and policy development. This is described in full in Sect. 6.2.1.

6.2.1 Implementing a tamper detection component

The tamper detection system of the REDE sensor nodes consists of a single LooCI component, which aggregates sensor readings from the internal 3D accelerometer and an associated filtering component.

The *Accelerometer* component provides a single interface that publishes a scalar reading for total acceleration across all axes. The *Accelerometer* policy filters these readings at the source node, only allowing remote publication where the value exceeds a predefined threshold. In this way, the filter policy prevents the publication of spurious alerts due to movement caused by wind and heavy rain and thus conserves battery power. A complete listing for this component is provided in Listing 3 which shows how sensor readings may be disseminated over the event bus using LooCI.

In order to better understand the effect of environmental conditions on sensor node movement, we monitored the output of the *Accelerometer* component on a deployed REDE sensor node at the test site described in Sect. 3.2 for 24 hours during inclement weather. Readings from the *Accelerometer* component were recorded at one second intervals throughout the evaluation period. To simulate tampering with the REDE sensor node, we then had five volunteers attempt to open the sensor node enclosure three times in any way they chose, recording the output of the *Accelerometer* component in each case.

Figure 13 shows the maximum *Accelerometer* reading generated by natural movement of the REDE sensor nodes, along with the most successful attempt (i.e., the attempt with the lowest *Accelerometer* reading) of each volunteer who attempted to open the sensor node enclosure. It can be seen

Table 1 Minimum sleep periods required for continual operation

		<i>Required Sensing Period (Seconds)</i>									
		1	2	3	4	5	6	7	8	9	10
<i>Length of Required Transmission Period (Seconds)</i>	1	18	32	46	60	74	88	102	115	129	143
	2	22	37	50	64	78	92	106	120	134	148
	3	27	41	55	69	83	97	111	124	138	152
	4	32	46	60	73	87	101	115	129	143	157
	5	36	50	64	78	92	106	120	133	147	161
	6	41	55	69	83	96	110	124	138	152	165
	7	45	59	73	87	100	115	129	142	156	170
	8	50	64	78	92	105	119	133	147	161	175
	9	55	68	82	96	110	124	138	152	165	179
	10	59	73	87	100	114	128	142	156	170	184

Listing 3 The TamperDetection component

```

IEventPublisher {
    private byte[] publications =
        {ApplicationEventTypes.THEFT_READING};
    private IAccelerometer3D vibrationSensor =
        EDemoBoard.getInstance().getAccelerometer();
    protected void startApp() throws MIDletStateChangeException {
        initialise(); // Connect to the LooCI run-time
        while (getState() == STARTED) { // Send readings over Event Bus
            PayloadBuilder payload = new PayloadBuilder();
            double x = vibrationSensor.getAccelX();
            double y = vibrationSensor.getAccelY();
            double z = vibrationSensor.getAccelZ();
            double totalVibration = x + y + z;
            int total = (int)totalVibration;
            payload.addInteger(total);
            publishEvent(new Event(ApplicationEventTypes.THEFT_READING,
                payload.getPayload()));
            Thread.sleep(Parameterization.getTiming());
        }
    }
    public byte getComponentID() { // Return the component type
        return ApplicationComponentTypes.THEFT_SENSOR; }
    public byte[] getPublications() { // Return the interface types
        return publications; }
} // End of class VibrationSensor

```

that for every case, attempts to open the sensor node enclosure resulted in a higher *Accelerometer* reading than for natural movement.

It is important to note that we do not expect the above experiments to provide a fool-proof method for distinguishing between natural movement and movement caused by tampering with the REDE sensor nodes. For this reason, we choose to implement the filtering of *ACCEL* events (see

Fig. 5) using a PMA-based policy that can be easily tailored at run-time. A complete listing for this policy is provided in Listing 4.

As can be seen from Listing 4, the tamper detection policy is significantly more compact than the *Accelerometer* component, and thus if the need to reconfigure the system arises (for example to adapt to changing levels of natural

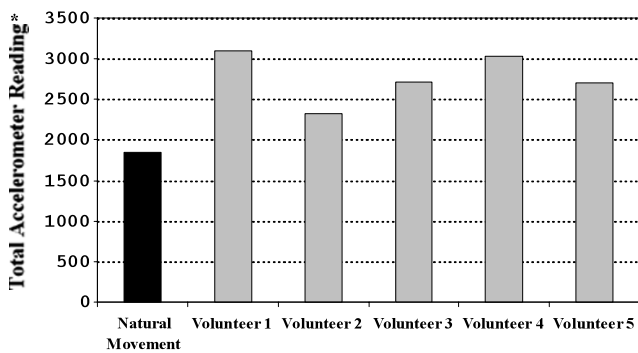


Fig. 13 Output from the accelerometer component. (*The output of this component is the scalar sum of the SunSPOT accelerometer on three axes (an ADC value of between 0 and 1023) and, therefore, ranges between 0 and 3069)

```
policy "tamper detection" {
  on ACCELEROMETER_SENSOR_READING_EVENT as a;
  if( a.value >= 2000 ) then (
    publish SENSOR_READING_EVENT(a.eventID,
      a.src_comp, a.value); )
}
```

Listing 4 The accelerometer filtering policy

movement) the distribution of policy updates imposes minimal energy overhead.

6.3 Application footprint

For resource constrained embedded systems, maintaining a minimal application footprint is critical for two primary reasons: firstly, a small application footprint leaves more space available for data logging. Secondly, in the case of reconfigurable systems, where component updates may be transmitted over the air, a smaller component footprint reduces radio activity, and thus conserves battery life.

The *static memory footprint* of all components in our application composition is shown in Table 2. As can be seen from the table, all components have a footprint of less than 2 kB, with the exception of The *EnvironmentalHazardAlert* component that runs on the resource-rich back-end and, therefore, its larger size has no significant impact on system performance.

The footprint of all policies in our application composition is shown in Table 3. As can be seen from the table, policies are much more compact than components, with all policies consuming between 94 bytes and 95 bytes—an order of magnitude less than components. This is particularly advantageous as policies are used to represent dynamic concerns such as alert thresholds and are therefore more likely to be updated at run-time over-the-air. The *dynamic memory requirements* of each component vary depending on us-

Table 2 Component footprint

	Size
ConductivitySensor	1.8 kB
MethaneSensor	1.7 kB
PressureSensor	1.7 kB
AccelerometerSensor	1.9 kB
EnvironmentalHazardAlert	13.2 kB
Total	20.3 kB

Table 3 Policy footprint

	Size
ConductivityDetectionPolicy	94 bytes
MethaneDetectionPolicy	94 bytes
TamperDetectionPolicy	95 bytes
FloodDetectionPolicy	94 bytes
Total	377 bytes

Table 4 Lines of code: components

	Functional code	LooCI code
ConductivitySensor	48	11
MethaneSensor	48	11
PressureSensor	48	11
AccelerometerSensor	42	11
EnvironmentalHazardAlert*	41	23
Total	227	67

*LoC for the supporting GUI is not included in this analysis, as it is unrelated to the application

age context, however, each policy consumes approximately 400 bytes of RAM and each microcomponent consumes approximately 2 kB of RAM.

6.4 Development overhead

This section provides a quantitative assessment of development overhead. Tables 4 and 5, respectively, provide a Lines of Code (LoC) analysis for all components and policies in our application. As can be seen from Table 4, LooCI components are relatively compact and impose limited overhead on developers in terms of LooCI-specific code. The identical size of the first three components listed in the table may be attributed to their simplicity and similarity (in all cases, these components read a simple value from the Sun SPOT Analogue to Digital Converter (ADC) and transmit it over the event bus).

Table 5 Lines of code: policies

	Size
ConductivityDetectionPolicy	10
MethaneDetectionPolicy	10
TamperDetectionPolicy	10
FloodDetectionPolicy	10
Total	40

As shown in Table 5, the high-level nature of PMA policies allows compact expression of rich functional and non-functional objectives.

7 Future work

Our initial future work will focus on realising a large scale deployment of REDE sensor nodes to monitor river conditions on the Tiete River in the city of São Paulo. This work will allow us to analyse the scalability of the LooCI event bus communication abstraction [27] and better evaluate the efficiency of our policy-based management architecture [17]. In addition to providing interesting technical data, a REDE deployment in the city of São Paulo will serve the public interest by issuing warnings of impending flood and pollution events. In the extended Tiete River deployment, power efficient Contiki-based sensor nodes such as the AVR Raven [8] will be used to support simple sensors, while powerful sensor nodes such as the Sun SPOT [41] will provide additional functionality such as encryption and logging. Using this heterogeneous scenario, we will investigate the benefits of LooCI as an interoperability layer that can integrate heterogeneous sensor node platforms.

Building upon our large-scale deployment on the River Tiete, we will undertake a significant evaluation of the effectiveness of REDE in providing timely and reliable flood warnings. Specifically, this evaluation will assess the trade-off between timeliness and reliability and attempt to isolate the optimum lead time for warning distribution.

In tandem with the research directions outlined above, we will also work to build a suite of tools that will facilitate the development, management, and use of DisSeNT applications. In particular, we will develop an integrated development environment for the multiparadigm programming approach used in this scenario. This development environment will provide support for high-level application composition using QARI [28], generic component development using LooCI [27] and policy-based management using PMA [17]. We will also develop a suite of graphical tools for the River Monitoring application presented in this paper.

8 Conclusions

This paper has explored how wireless sensor network technologies can be used to realise a next-generation river monitoring system. We have designed and evaluated a custom hardware platform called REDE that meets the specific requirements of river monitoring and warning in São Paulo state. In addition to the REDE hardware platform, we have demonstrated how the multiparadigm programming approach offered by DisSeNT can be used to build efficient, adaptable and maintainable applications.

Evaluation has shown that our custom hardware platform provides flexible sensing and low-power consumption. In conjunction with the solar array, the low power consumption of REDE sensor nodes ensures continuous operation without the need for battery replacement. Furthermore, applications developed using DisSeNT are efficient in terms of both lines-of-code and memory footprint. Together these features make the developed system a good match with the requirements of river monitoring in the state of São Paulo.

In a broader sense, it is our belief that the multiparadigm programming approach embodied in DisSeNT holds great potential for large scale multiactor scenarios such as the river monitoring application described in this paper. DisSeNT respects both the skill-set of each actor and the resource constraints of WSN environments.

Acknowledgements This research is conducted in the context of the IWT-STADIUM project No. 80037 [47]. Wouter Horré is a Ph.D. fellow of the Research Foundation—Flanders (FWO).

Dr. Danny Hughes and Dr. Jo Ueyama would like to thank FAPESP for funding the case-study elements of this research project (Process ID 08/05346-4 and 09/01881-5).

References

1. Douglas BC (1997) Global sea rise: a redetermination. *Surv Geophys* 18:279–292
2. Mendiondo EM (2008) Challenging issues of urban biodiversity related to ecohydrology. *Braz J Biol* 68(4):983–1002
3. Römer K, Mattern F (2004) The design space of wireless sensor networks. *IEEE Wirel Commun* 11(6):54–61
4. Mainwaring A, Polastre J, Szewczyk R, Anderson J (2002) Wireless sensor networks for habitat monitoring. In: *Proc of 1st ACM international workshop on wireless sensor networks and applications*, Atlanta, Georgia, USA, pp 88–97
5. Hughes D, Greenwood P, Coulson G, Blair G, Pappenberger F, Smith P, Beven K (2007) An experiment with reflective middleware to support grid-based flood monitoring. *Inter-Sci J Concurr Comput, Pract Exp* 20(11):1303–1316
6. Taneja J, Jeong J, Culler D (2008) Design, modelling, and capacity planning for micro-solar power sensor networks. In: *Proc of the 7th international conference on information processing in sensor networks, IPSN SPOTS '08*, pp 407–418
7. Hill J, Szewczyk R, Woo A, Hollar S, Culler D, Pister K (2000) System architecture directions for networked sensors. *ACM SIGPLAN* 35(11):93–104

8. Dunkels A, Grönvall B, Voigt T (2004) Contiki—a lightweight and flexible operating system for tiny networked sensors. In: Proc of 29th IEEE international conference on local computer networks, LCN'04, Tampa, FL, USA, pp 455–462
9. Han CC, Rengaswamy RK, Shea R, Kohler E, Srivastava M (2005) SOS: A dynamic operating system for sensor networks. In: Proc of the 3rd international conference on mobile systems, applications and services, Mobisys'05, pp 211–224
10. IEEE Computer Society (2009) IEEE Standard 802.15.4d-2009, April 17
11. Crossbow, MICA-Z wireless measurement system. Data sheet http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAZ_Datasheet.pdf
12. MoteIV. T-MOTE sky ultra-low power wireless module. Data sheet <http://www.cs.uvm.edu/~crobinso/mote/tmote-sky-datasheet-102.pdf>
13. DeRoure D. Improving flood warning times using pervasive and grid computing. Available online at <http://envisense.org/floodnet/ingenia/ingenia.htm>
14. Microsystems Sun Java ME—the most ubiquitous application platform for mobile devices. <http://java.sun.com/javame/index.jsp>
15. Sentilla. Perk frequently asked questions. http://www.sentilla.com/perk_faq.html
16. Coulson G, Blair G, Grace P, Taiani F, Joolia A, Lee K, Ueyama J, Sivaharan T (2008) A generic component model for building systems software. *ACM Trans Comput Syst* 26(1)
17. Matthys N, Huygens C, Hughes D, Michiels S, Joosen W (2009) Flexible integration of data qualities in wireless sensor networks. In: Proc of the 4th international workshop on middleware tools, services and run-time support for sensor networks, MidSens'09, Urbana Champaign, Illinois, USA
18. Waysmall Computers. Gumstix embedded computing platform specifications. Available online at <http://gumstix.com/spexboards.html>
19. Costa P, Coulson G, Gold R, Lad M, Mascolo C, Mottola L, Picco GP, Sivaharan T, Weerasinghe N, Zachariadis S (2007) The RUNES middleware for networked embedded systems and its application in a disaster management scenario. In: Proc of the 5th annual IEEE international conference on pervasive computing and communications, PerCom'07, White Plains, New York, pp 69–78
20. Porter B, Coulson G (2009) Lorien: a pure dynamic component-based operating system for wireless sensor networks. In: Proc of the 4th international workshop on middleware tools, services and run-time support for sensor networks, MidSens'09, Urbana Champaign, Illinois, USA
21. Russello G, Mostarda L, Dulay N (2008) Escape: a component-based policy framework for sense and react applications. In: Proc of the 11th international symposium on component-based software engineering, CBSE'08, pp 212–229
22. Zhu Y, Keoh S, Sloman M, Lupu E (2009) A lightweight policy system for body sensor networks. *IEEE Trans Netw Serv Manag* 6(3):137–148
23. Huygens C, Joosen W (2009) Federated and shared use of sensor networks through security middleware. In: Proc of the 6th international conference on information technology: new generations, ITNG'09, Las Vegas, Nevada, USA, pp 1005–1011
24. Steffan J, Fiege L, Cilia M, Buchmann A (2005) Towards multi-purpose wireless sensor networks. In: Proc of IEEE systems communications, ICW'05, Montreal, Canada, pp 336–341
25. Iwao T, Nomura K, Pitt J, Amamiya M (2005) A control model of multi-purpose sensor networks by policies. In: Proc of active media technology, AMT'05, Kagawa, Japan, pp 429–434
26. Kansal A, Nath S, Liu J, Zhao F (2007) SenseWeb an infrastructure for shared sensing. *IEEE Multimed* 14(4):8–13
27. Hughes D, Thoelen K, Horré W, Matthys N, Michiels S, Huygens C, Joosen W (2009) LooCI: a loosely-coupled component infrastructure for networked embedded systems. In: Proc of the 7th international conference on advances in mobile computing & multimedia, MoMM'09. ACM, New York
28. Horré W, Hughes D, Michiels S, Joosen W. QARI: Quality aware software deployment for wireless sensor networks. In: Proc of the 7th international conference on information technology: new generations, ITNG'10
29. Mendiondo EM, Valdes JB (2002) Strategies for sustainable development of water resources systems. In: Proc of 2nd int conf new trends in water & environ eng for safety & life: eco-compatible solutions for aquatic ecosystems, Capri, Italy, 2002
30. Mendiondo EM (2005a) Scenarios of South American floods—From mitigating disasters to early-warning strategies In: Takara K et al (eds) Proceedings from international conference on monitoring. Prediction and mitigation of water-related disasters MPMD 2005, Kyoto University, Kyoto, 12–15 Jan. Kyoto: DPRI
31. Mendiondo EM (2010) Reducing vulnerability to water-related disasters in urban areas of the humid tropics. In: Parkinson et al (eds) Integrated urban water management: humid tropics. Urban water series, vol 6. Leiden: UNESCO/Taylor&Francis, pp 109–124. Chapt. 6, ISSN 1749-0790
32. Clarke RT, Mendiondo EM, Brusa LC (2000) Uncertainties in mean discharges from two large South American rivers due to rating curve variability. *Hydrol Sci J* 45(2):221–236
33. Neiff JJ, Mendiondo EM, Depettris CA (2000) ENSO floods on river ecosystems: from catastrophes to myths. In: Toensmann F, Koch M (eds) “River flood defence”, Proceedings from I international symposium on flood defence. Kassel: Herkules
34. Tucci CEM (2010) Integrated urban water management in the humid tropics. In: Parkinson et al (eds) Integrated urban water management: humid tropics. Urban Water Series, vol 6. Leiden: UNESCO/Taylor&Francis, pp 1–15. Chapt. 1, ISSN 1749-0790
35. Benetti AD, Campos L (2010) Water supply and wastewater management in the humid tropics. In: Parkinson et al (eds) Integrated urban water management: humid tropics. Urban Water Series, vol 6. Leiden: UNESCO/Taylor&Francis, pp 25–40. Chapt. 2, ISSN 1749-0790
36. Mendiondo E (2008) Challenging issues of urban biodiversity related to ecohydrology. *Braz J Biol* 68:983–1002
37. Clarke T, Mendiondo E, Brusa L (2000) Uncertainties in mean discharge from two South American rivers (Amazonas and Paraná) due to rating curve variability. *Hydrol Sci J* 45(2):221–236
38. Barros R, Wendland E, Mendiondo E (2007) Cálculo de áreas inundáveis devido às enchentes para o Plano Diretor de Drenagem Urbana na Bacia Escola do Gregório, São Carlos. *Rev Bras Recur Hídric* 12:5–17
39. Coulson G, Hughes D, Blair G, Grace P (2008) The evolution of the GridStix wireless sensor network platform. In: Proc of the international workshop on sensor network engineering, IWSNE'08, Santorini, Greece
40. The LooCI project on google code. Available online at <http://code.google.com/p/looci/>
41. Simon D, Cifuentes C, Cleal D, Daniels J, White D (2006) Java on the bare metal of wireless sensor devices: the squawk java virtual machine. In: Proc of the 2nd international conference on virtual execution environments, Ottawa, Canada, pp 78–88
42. Preuveneers D, Berbers Y (2008) Encoding semantic awareness in resource-constrained devices. *IEEE Intell Syst* 23(2):26–33
43. Smith P, Hughes D, Beven K, Cross P, Tych W, Coulson G, Blair G (2009) Towards the provision of site specific flood warnings using wireless sensor networks. *Inter-Sci J Meteorol Appl* 16(1):57–64
44. Prayatib A, Antonopoulousa C, Stoyanova T, Koulamas B, Papadopolousa G (2010) A modelling approach on the TelosB WSN platform power consumption. *J Syst Softw* 83(8):1355–1363
45. Sundaresan S, Koren I, Koren Z, Krishna CM (2009) Event-driven adaptive duty-cycling in sensor networks. *Int J Sens Netw* 6(2):89–100

46. Vigorito CM, Ganesan D, Barto AG (2007) Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In: Proc of 4th annual IEEE communications society conference on sensor, mesh and ad hoc communications and networks, SECON'07, San Diego, California, pp 21–30
47. IWT Stadium project 80037. Software technology for adaptable distributed middleware. <http://distrinet.cs.kuleuven.be/projects/stadium/>
48. Mosquera-Machado S, Ahmad S (2007) Flood hazard assessment of Atrato river in Colombia. *Water Resour Manag* 21(3):591–609