Baltic J. Modern Computing, Vol. 2 (2014), No. 2, pp. 84–115

Wireless Sensor Network Software Design Rules

Girts Strazdins[1,2], Leo Selavo[1,2]

[1] Faculty of Computing, University of Latvia
Raina blvd 19, Riga, LV 1050, Latvia
[2] Institute of Electronics and Computer Science
Dzerbenes 14, Riga LV 1006, Latvia

girts@strazdins.lv, leo.selavo@gmail.com

Abstract. In the last decade wireless sensor networks (WSNs) have evolved as a promising approach for smart investigation of our planet, providing solutions for environment and wild animal monitoring, security system development, human health telemonitoring and other domains. Lack of unified standards and methodologies leads to limited sensor network solution interoperability and portability. The goal of this work is to propose wireless sensor network software development design rules that serve as a unified methodology for operating system and application development. The design rules are based on 40 existing WSN deployment extensive analysis and common trend inference. Improvements for existing WSN deployments and operating systems are identified. The evaluation shows the proposed design rules as an important tool for WSN software development at different stages, from planning to testing and change request analysis.

Keywords: wireless sensor networks, methodology, design rules, operating systems, deployment survey, case study

1 Introduction

Environmental scientists, biologists, geologists and other researchers and industry professionals are interested in measuring a variety of parameters and phenomena of our planet. Wireless sensor networks (WSNs) is a paradigm of measuring and event detection in the surrounding environment. It is a tool for smart sensing of our planet, having wide variety of applications, including wild animal monitoring (Zhang et al., 2004), remote island flora inspection (Selavo et al., 2007), volcano eruption prediction (Werner-Allen et al., 2006), interactive dance music generation (Aylward and Paradiso, 2007), restricted area monitoring (Wittenburg et al., 2007), and battlefield surveilance (Arora et al., 2004), among others.

Sensor nodes typically are match-box size embedded devices (Figure 1). Such devices are called *motes*. Other networks of devices with sensing and communication capabilities can also be considered sensor networks, for example vehicular sensor networks with car on-board computers (Mednis et al., 2010) or human-centric networks of

smartphones (Burke et al., 2006). However, these kinds of networks use devices significantly different from motes, and therefore different software abstractions could be more appropriate. In this work, the main focus is on networks consisting of *mote* devices.

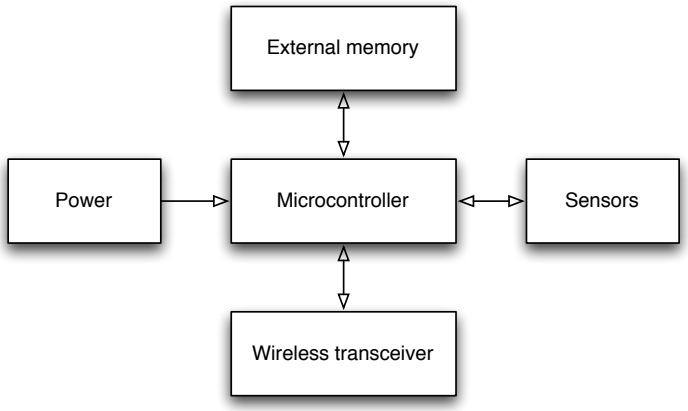

Fig. 1. A typical wireless sensor node architecture - power, microcontroller, wireless transceiver, sensors and external memory (optional)

Sensor network design and programming contains a set of challenges to solve:

1. Limited energy budget and energy efficiency is the main challenge of sensor networks. High-capacity batteries and energy harvesting methods (Kansal et al., 2007) have to be combined in an efficient way, using energy buffering and duty cycling.
2. Small size and low-power computation (see Table 1) implies limited central processing unit (CPU), random-access memory (RAM) and flash memory resources, which require highly effective software abstractions and low-level hardware control.
3. Standard communication hardware and protocol stacks used in Internet servers and desktop computers are not suitable for sensor networks, due to low-power requirements and duty cycling.
4. Different platforms are often used during evolution of the projects. System design requires rapid prototyping, including hardware and software tool support.
5. Embedded systems are event-driven, while most desktop programming languages provide sequential programming paradigms. Therefore easily-adoptable software abstractions must be provided to WSN programmers, without requirement of long learning process.
6. Sensor nodes are often deployed in the wild, industrial or other open and harsh environment, requiring additional effort for packaging problem solution.
7. Wireless communication has remarkable irregularities and disturbances, which must be mitigated by both hardware and software methods.

8. Despite the unfriendly environment, sensor networks should be fault tolerant and self-adaptive.
9. Deployment site is often hardly reachable for physical hardware inspection and maintenance. Remote real-time management support is therefore desirable.

Table 1. Typical mote resource limits - CPU performance and memory amounts are serious constraints for software design

Resource	Typical amount
CPU	1 - 20MHz
RAM memory	256B - 10KiB
Non-volatile flash memory	40 - 128KiB
Optional external flash memory	1MiB - 4GiB

Wireless sensor network research has evolved over the last decade and has reached a state where standardization becomes essential for interoperability between different hardware and software solutions. Although parts of the WSN solutions are standardized, such as communication protocols (802.15.4 standard (IEEE, 2014)), a common methodology for WSN software development is still missing. WSN designers face typical problems during software development.

Therefore central thesis of this work states: *a common methodology is required for wireless sensor network software development*. Such methodology would foster efficient new solution design and serve as a tool for existing software evaluation and identification of improvements.

The rest of this work describes the process of identification of common WSN problems, WSN design rule proposal and evaluation in different aspects: existing software assessment and new software design. These rules form a basis for common WSN software development methodology. Although standardization is a slow and complex process, these rules can have impact towards establishment of common WSN software development standards and protocols.

The main contribution in this work includes:

1. Analysis of 40 sensor network deployments described in the research literature. As a result the critical and recurring WSN properties were distilled.
2. Identification of common WSN design problems that identify the challenges based on critical WSN properties and user requirements.
3. Introduction of a WSN software development methodology in the form of 25 design rules and analysis of their mapping to underlying problems.
4. Evaluation of the proposed design rule impact on existing WSN software improvement. Design rules are shown as a tool for existing system comparison, drawback identification and future direction sketch. The evaluation consists of three parts:
   (a) Improvements to the analyzed deployment set showing design rule applicability in general, for WSN users.

(b) Existing operating system conformance to proposed rules and suggestions for OS improvement. Design rules are shown as an important tool for WSN OS developers. This evaluation includes the authors' participation in the development and improvement analysis of MansOS: a portable operating system (OS) for sensor networks.
(c) A wearable sensor network use-case scenario - assessment of prototype implementation and suggestions for future work. This part shows more detailed improvement of a particular WSN deployment in terms of network lifetime and network coverage.

2 Related work

For WSN requirement summary and trend inference we first have to survey existing WSNs and establish a taxonomy. Numerous researchers have surveyed and described sensor network characteristics and challenges. In her book Anna Hac describes sensor networks in general, including typical challenges (Hac, 2003). Hill et.al. describe WSN hardware platforms (Hill et al., 2004). Tilak et al. propose to categorize sensor networks based on different criteria (Tilak et al., 2002). Mottola and Picco propose another taxonomy focusing on programming aspects (Mottola and Picco, 2011). During this work authors have also developed a taxonomy, described in more detail in (Strazdins, 2014). The taxonomy is based on the authors' experience and summary of multiple survey articles.

Metric definition is also an important task. Beutel proposes metrics for WSN hardware platforms (Beutel, 2006). Here we used subset of these metrics for deployment analysis in Section 4. However, we add significantly more metrics in the survey, see full list of tables in (Strazdins, 2014).

Romer and Mattern have analyzed WSN deployments and assessed the WSN design space based on application characteristics (Romer and Mattern, 2004). This work includes similar deployment analysis approach. Handziski et.al. have analyzed WSN challenges and come to conclusions similar to the authors': standardizations and unified methodologies are required (Handziski et al., 2003). Handzinski is proposing suggestions for handling the challenges, without formalization. Jason Hill's thesis (Hill, 2003) is the closest effort to this paper. He analyzes WSN system architecture, describes constraints and challenges. Hill substantiates TinyOS design choices with qualitative suggestions based on the identified challenges. We take a step further and propose specific design rules based on a deployment survey. The WSN survey is performed similarly to previous work, yet with more detail and more formalized outcomes: the proposed design rules.

Different approaches are possible in WSN design and specification. Several researchers have proposed algorithms and formulas to optimize WSN communication. Mhatre and Rosenberg propose an algorithm how to choose between different communication approaches (Mhatre and Rosenberg, 2004). Olariu and Stojmenovic describe formulas how to calculate energy depletion dependence on network topology and optimal transmission power (Olariu and Stojmenovic, 2006). Stojmenovic et.al. describe design guidelines for WSN routing protocols (Stojmenovic et al., 2005). Oppermann

and Peter propose a framework to transform informal end-user requirements to technical specifications (Oppermann and Peter, 2010). It tries to solve communication problem between different WSN user groups: end-users and engineers.

In contrast, we focus on optimizing software development process, not communication protocols or social communication problems. We extract technical WSN deployment characteristics based on the information available. In some cases it is not possible to gather quantitative information. Qualitative discussion is used in such situations. Nevertheless, to the best of the authors' knowledge, this work proposes the most comprehensive and formalized set of design rules for WSN software development.

3 Software abstractions

Sensor nodes can be programmed using different abstractions (Figure 2). The choice of abstraction to use is up to user and depends on application requirements and user skills.

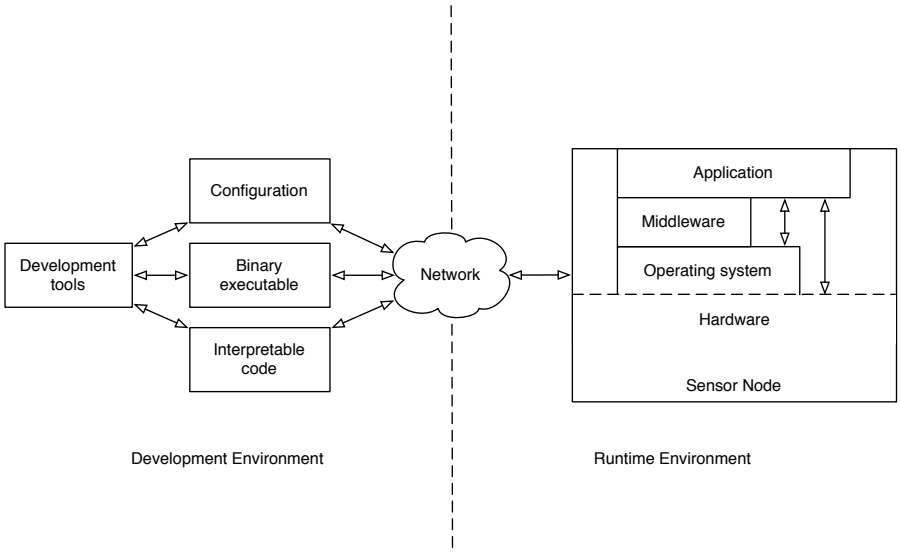

Fig. 2. Abstractions for sensor networks - Application can be written using custom code, on-top of operating system or using additional middleware. The whole network can be reprogrammed or re-tasked remotely

The following abstractions are typical in runtime environment:

- Direct hardware access in the application user creates the whole application.
- Operating system provides ready-to-use services and libraries, maintaining efficient low-level hardware access.
- Middleware additional services and interfaces for simplified and generalized access, losing some degree of expressiveness.

– Application user configures and adapts template applications.

This paper concentrates on the operating system level, as it is a efficient tradeoff between flexible access and expressiveness, and satisfactory programming complexity, not requiring long training. In addition, application level is also reviewed and middleware level is summarized.

The following WSN operating systems are analyzed in this paper:

– MansOS portable and easy to use WSN operating system (Strazdins et al., 2010; Elsts et al., 2012).
– TinyOS de facto standard in the WSN community, however, has a very steep learning curve (Levis et al., 2005).
– Contiki provides rich service set and simple programming interface, yet its source code is less portable, compared to other solutions (Dunkels et al., 2004).
– LiteOS brings multiple Unix concepts to sensor networks; AVR architecture- specific, not portable (Cao et al., 2008).
– Mantis provides signal processing algorithms and multiple platform support; not maintained; MCU and architecture code mixed in source files (Bhatti et al., 2005).
– Arduino development environment for electronics enthusiasts, targeted for embedded system prototyping, can be adapted to sensor network needs (Arduino SA, 2014a).

4 Deployment survey

To understand WSN application trends, the authors have performed WSN deployment survey. 40 research papers published in 2002-2011 have been analyzed, covering wide WSN application range (see Table 2). In contrast to previously published deployment surveys this survey describes not only WSN deployments in stable phase, yet also pilot projects and prototypes, as requirements for rapid and dynamic software development and debugging tools is essential in these stages. Full details are described in Strazdins' PhD thesis Strazdins (2014). The main conclusions are summarized here:

– In 80% of cases network consists of less than 50 nodes; in 50% of cases lesS than 20 nodes; in 34% - 10 or fewer nodes. Therefore WSN software developers should focus on applicability and portability of the solutions, scalability is secondary.
– Off-the-shelf nodes are used only in 20% of deployments. In the remaining 80% nodes are either customized or fully custom-designed. 60% use TinyOS, however, 25% use custom OS. Easily portable software solutions are required, adaptable to every platforms needs.
– In 33% of cases application consists of a single central task. However, in more complex scenarios up to 6 parallel processes are executed.
– Preemptive scheduling (kernel forcedly switches control between user tasks on a periodic schedule) is required in 30% of cases. In the remaining 70% cooperative scheduling with explicit control yield is sufficient.

- Time synchronization is used in 38% of cases, therefore it should be included as a core service of the operating system. While also localization is used in 38% of deployments, it uses very different environmental conditions in each case (indoor/outdoor, with/without additional infrastructure), and it is not possible to implement universal localization service.
- Temperature, light and acceleration sensors are most widely used. Most of sensors are analog; analog-to-digital converter (ADC) is used to sample them.
- Significant part of deployments (40%) have target lifetime of 7 days or more, therefore energy-effective software solutions are required.
- In more than 80% of cases network contains nodes with continuous power source. This fact can be used to simplify network protocols.
- The most popular hardware platform in recent years is TelosB (TMote Sky). Popular microcontroller architectures: TI MSP430 and Atmel AVR. 802.15.4- compatible radio chips are widely used, including TI CC2420.
- Multi-hop networking is used in 65% of deployments.
- 70% use Carrier Sense Multiple Access (CSMA) based Media Access Control (MAC) protocols.
- Remote data access and node reprogramming is used in 35% of cases.

These findings form a basis for typical WSN problem and design rule definition in Sections 5 and 6.

Table 2: Deployments: general information

Nr	Codename	Year	Title	Class	Description
1	Habitats (Mainwaring et al., 2002)	2002	Wireless sensor networks for habitat monitoring	Habitat and weather monitoring	One of the first sensor network deployments, designed for bird nest monitoring on a remote island
2	Minefield (Merrill et al., 2003)	2003	Collaborative Networking Requirements for Unattended Ground Sensor Systems	Opposing force investigation	Unattended ground sensor system for self healing minefield application
3	Battlefield (He et al., 2004)	2004	Energy-Efficient Surveillance System Using Wireless Sensor Networks	Battlefield surveillance	System for tracking of the position of moving targets in an energy-efficient and stealthy manner
4	Line in the sand (Arora et al., 2004)	2004	A line in the sand: a wireless sensor network for target detection, classification, and tracking	Battlefield surveillance	System for intrusion detection, target classification and tracking
5	Countersniper (Simon et al., 2004)	2004	Sensor Network-Based Countersniper System	Opposing force investigation	An ad-hoc wireless sensor network-based system that detects and accurately locates shooters even in urban environments.
6	Electroshepherd (Thorstensen et al., 2004)	2004	Electronic shepherd - a low-cost, low-bandwidth, wireless network system	Domestic animal monitoring and control	Experiments with sheep GPS and sensor tracking
7	Virtual fences (Butler et al., 2004)	2004	Virtual fences for controlling cows	Domestic animal monitoring and control	Experiments with virtual fence for domestic animal control

...

Table 2 – continued

Nr	Codename	Year	Title	Class	Description
8	Oil tanker (Krishnamurthy et al., 2005)	2005	Design and Deployment of Industrial Sensor Networks: Experiences from a Semiconductor Plant and the North Sea	Industrial equipment monitoring and control	Sensor network for industrial machinery monitoring, using Intel motes with Bluetooth and high-frequency sampling
9	Enemy vehicles (Sharp et al., 2005)	2005	Design and Implementation of a Sensor Network System for Vehicle Tracking and Autonomous Interception	Opposing force investigation	A networked system of distributed sensor nodes that detects an evader and aids a pursuer in capturing the evader
10	Trove game (Mount et al., 2005)	2005	Trove: a Physical Game Running on an Ad-Hoc Wireless Sensor Network	Child education and sensor games	Physical multiplayer real-time game, using collaborative sensor nodes
11	Elder RFID (Ho et al., 2005)	2005	A Prototype on RFID and Sensor Networks for Elder Healthcare: Progress Report	Medication intake accounting	In-home elder healthcare system integrating sensor networks and RFID technologies for medication intake monitoring
12	Murphy potatoes (Langendoen et al., 2006)	2006	Murphy Loves Potatoes Experiences from a Pilot Sensor Network Deployment in Precision Agriculture	Precision agriculture	A rather unsuccessful sensor network pilot deployment for precision agriculture, demonstrating valuable lessons learned
13	Firewxnet (Hartung et al., 2006)	2006	FireWxNet: A Multi-Tiered Portable Wireless System for Monitoring Weather Conditions in Wildland Fire Environments	Forest fire detection	A multi-tier WSN for safe and easy monitoring of fire and weather conditions over a wide range of locations and elevations within forest fires
14	AlarmNet (Wood et al., 2006)	2006	ALARM-NET: Wireless Sensor Networks for Assisted-Living and Residential Monitoring	Human health telemonitoring	Wireless sensor network for assisted-living and residental monitoring, integrating environmental and physiological sensors and providing end-to-end secure communication and sensitive medical data protection
15	Ecuador Volcano (Werner-Allen et al., 2006)	2006	Fidelity and Yield in a Volcano Monitoring Sensor Network	Volcano monitoring	Sensor network for volcano seismic activity monitoring, using high frequency sampling and distributed event detection
16	Pet game (Liu and Ma, 2006)	2006	Wireless Sensor Network Based Mobile Pet Game	Child education and sensor games	Augmenting mobile pet game with physical sensing capabilities: sensor nodes act as eyes, ears and skin
17	Plug (Lifton et al., 2007)	2007	A Platform for Ubiquitous Sensor Deployment in Occupational and Domestic Environments	Smart energy usage	Wireless sensor network for human activity logging in offices, sensor nodes implemented as power strips
18	B-Live (Santos et al., 2007)	2007	B-Live - A Home Automation System for Disabled and Elderly People	Home/office automation	Home automation for disabled and elderly people integrating heterogeneous wired and wireless sensor and actuator modules

...

Table 2 – continued

Nr	Codename	Year	Title	Class	Description
19	Biomotion (Aylward and Paradiso, 2007)	2007	A Compact, High-Speed, Wearable Sensor Network for Biomotion Capture and Interactive Media	Smart user interfaces and art	Wireless sensor platform designed for processing multipoint human motion with low latency and high resolutions. Example applications: interactive dance, where movements of multiple dancers are translated into real-time audio or video
20	AID-N (Gao et al., 2007)	2007	The Advanced Health and Disaster Aid Network: A Light-Weight Wireless Medical System for Triage	Human health telemonitoring	Lightweight medical systems to help emergency service providers in mass casualty incidents
21	Firefighting (Wilson et al., 2007)	2007	A Wireless Sensor Network and Incident Command Interface for Urban Firefighting	Human-centric applications	Wireless sensor network and incident command interface for firefighting and emergency response, especially in large and complex buildings. During a fire accident, fire spread is tracked and firefighter position and health status is monitored.
22	Rehabil (Jarochowski et al., 2007)	2007	Ubiquitous Rehabilitation Center: An Implementation of a Wireless Sensor Network Based Rehabilitation Management System	Human indoor tracking	Zigbee sensor network based ubiquitous rehabilitation center for patient and rehabilitation machine monitoring
23	CargoNet (Malinowski et al., 2007)	2007	CargoNet: a low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events	Good and daily object tracking	System of low-cost, micropower active sensor tags for environmental monitoring at the crate and case level for supply-chain management and asset security
24	Fence monitor (Wittenburg et al., 2007)	2007	Fence Monitoring Experimental Evaluation of a Use Case for Wireless Sensor Networks	Security systems	Sensor nodes attached to a fence for collaborative intrusion detection
25	BikeNet (Eisenman et al., 2010)	2007	The BikeNet Mobile Sensing System for Cyclist Experience Mapping	City environment monitoring	Extensible mobile sensing system for cyclist experience (personal, bicycle and environmental sensing) mapping leveraging opportunistic networking principles
26	BriMon (Chebrolu et al., 2008)	2008	BriMon: A Sensor Network System for Railway Bridge Monitoring	Bridge monitoring	Delay tolerant network for bridge vibration monitoring using accelerometers. Gateway mote collects data and forwards opportunistically to a mobile base station attached to a train passing by.
27	IP net (Finne et al., 2008)	2008	Experiences from Two Sensor Network Deployments - Self-Monitoring and Self-Configuration Keys to Success	Battlefield surveillance	Indoor and outdoor surveillance network for detecting troop movement
28	Smart home (Suh et al., 2006)	2008	The Design and Implementation of Smart Sensor-based Home Networks	Home/office automation	Wireless sensor network deployed in a miniature model house, which controls different household equipment: window curtains, gas valves, electric outlets, TV, refrigerator, door locks

...

Table 2 – continued

Nr	Codename	Year	Title	Class	Description
29	SVATS (Song et al., 2008)	2008	SVATS: A Sensor-network-based Vehicle Anti-Theft System	Anti-theft systems	Low cost, reliable sensor-network based, distributed vehicle anti-theft system with low false-alarm rate
30	Hitchhiker (Barrenetxea et al., 2008)	2008	The Hitchhikers Guide to Successful Wireless Sensor Network Deployments	Flood and glacier detection	Multiple real-world sensor network deployments performed, including glacier detection, experience and suggestions reported.
31	Daily morning (Ince et al., 2008)	2008	Detection of Early Morning Daily Activities with Static Home and Wearable Wireless Sensors	Daily activity recognition	Flexible, cost-effective, wireless in-home activity monitoring system integrating static and mobile body sensors for assisting patients with cognitive impairments
32	Heritage (Ceriotti et al., 2009)	2009	Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment	Heritage building and site monitoring	Three different motes (sensing temperature, vibrations and deformation) deployed in a historical tower to monitor its health and identify potential damage risks.
33	AC meter (Jiang et al., 2009)	2009	Design and Implementation of a High-Fidelity AC Metering Network	Smart energy usage	AC outlet power consumption measurement devices, which are powered from the same AC line, but communicate wirelessly to IPv6 router
34	Coal mine (Li and Liu, 2009)	2009	Underground Coal Mine Monitoring with Wireless Sensor Networks	Coal mine monitoring	Self-adaptive coal mine WSN system for rapid detection of structure variations caused by underground collapses
35	ITS (Francescchinis et al., 2009)	2009	Wireless Sensor Networks for Intelligent Transportation Systems	Vehicle tracking and traffic monitoring	Traffic monitoring system implemented through WSN technology within SAFESPOT Project
36	Underwater (Detweiler et al., 2010)	2010	Adaptive Decentralized Control of Underwater Sensor Networks for Modeling Underwater Phenomena	Underwater networks	Measurement of dynamics of underwater bodies and their impact in the global environment, using sensor networks with nodes adapting their depth dynamically
37	PipeProbe (Lai et al., 2010)	2010	PipeProbe: A Mobile Sensor Droplet for Mapping Hidden Pipeline	Power line and water pipe monitoring	Mobile sensor system for determining the spatial topology of hidden water pipelines behind walls
38	Badgers (Dyo et al., 2010)	2010	Evolution and Sustainability of a Wildlife Monitoring Sensor Network	Wild animal monitoring	Badger monitoring in a forest
39	Helens volcano (Huang et al., 2012)	2011	Real-World Sensor Network for Long-Term Volcano Monitoring: Design and Findings	Volcano monitoring	Robust and fault-tolerant WSN for active volcano monitoring
40	Tunnels (Ceriotti et al., 2011)	2011	Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels	Tunnel monitoring	Closed loop wireless sensor and actuator system for adaptive lighting control in operational tunnels

5 Typical wireless sensor network problems

The author has identified common WSN problems that will be addressed by the proposed design rules. The problems are grouped and described in the following subsections.

5.1 Portability problems

Problem 1: **Chip reuse**. Various hardware platforms do exist that have common microchip and sensor base but different wiring and combination.

Problem 2: **Field experts**. WSN is a promising field not only for programmers and electrical engineers but also field experts with limited programming skills.

Problem 3: **Hardware evolution**. WSN users may choose different hardware platforms during the evolution of sensor network. However, the application logic and source code should be portable with minimal modifications.

5.2 Wireless communication problems

Problem 4: **Protocol variety**. Many WSN communication protocols do exist, yet not many ready-to-use implementations are available.

Problem 5: **WSN $\neq$ Internet**. WSN architecture is completely different from the Internet. Traditional protocols are not optimal, custom approach is required.

Problem 6: **Complex protocols**. Communication protocols are often too complex to provide full flexibility in unreliable networks.

Problem 7: **Limited resources**. WSNs must be able to communicate in dynamic topologies. However, memory and other resources are limited.

Problem 8: **Experimentation**. WSN researchers investigate and analyze protocols. An environment and infrastructure for experimentation is required.

Problem 9: **QoS**. A certain degree of Quality-of-Service (QoS) is required, especially in applications where WSNs are replacing traditional wired solutions.

5.3 Services and efficiency problems

Problem 10: **Energy**. The central problem of WSNs is energy efficiency, yet many pilot and prototype deployments use 100% duty cycle. Such approach may incur significant loss of realism in these deployments.

Problem 11: **Data caching**. Dynamic Networks with probabilistic communication may require data caching and preprocessing. In addition, local data logging for redundancy might be important, especially during prototyping phases.

Problem 12: **Complex states**. It is hard for programmers to think in event-driven approach that requires explicit management of system state and split-phase operation. Also it is hard to design programs with multiple concurrent events within a single thread of execution.

Problem 13: **Cooperation**. The whole network of nodes should cooperate to reach the real benefits of WSNs: higher resolution, energy efficiency and cooperative decision making.

6 WSN design rules

The following subsections list WSN design rules proposed by the authors. Although most of the rules are related to operating systems, they can be generalized to other WSN software abstractions: middleware and applications. Design rules in the form operating system should provide X become should use X in the application level. Detailed substantiation for each rule can be found in the full text of Strazdins thesis Strazdins (2014). Only definitions are listed in this paper.

The importance of rules is divided into the following classes based on their popularity among analyzed deployments:

1. MUST: the feature must be implemented. Deployment support: 40-100%.
2. SHOULD: feature implementation has lower impact. Deployment support: 20-50%.

Feature classes are overlapping in terms of popularity in deployments, as it is hard to define sharp thresholds for feature popularity and strictly assign importance classes. Overlapping boundaries open space for discussion.

Proposed rules are divided into multiple categories based on addressed aspects of WSN development. Table 3 lists all proposed rules in a summarized form.

Table 3: WSN software design rules proposed by the author

#	Rule	Description	Importance
		Communication	
1	Sink-oriented	The provided communication protocols must be sink-oriented.	MUST
2	Powered motes	Powered mote availability must be considered when designing a default networking protocol library.	MUST
3	30 byte payload	Default packet size should be at least 30 bytes with option to change this constant easily, when required.	SHOULD
4	11 hop routing	Multi-hop routing must be provided as a default component, which can be turned off, if one-hop topology is used. Topology changes must be expected, at least 11 hops should be supported.	MUST
5	CSMA MAC	A simple and generic CSMA-based MAC protocol must be included in WSN solutions, preferable as part of OS libraries.	MUST
6	Custom protocol API	Interface for custom MAC and routing protocol development must be provided.	MUST
7	Packet acknowledgment	Simple transport layer delivery acknowledgment mechanisms must be provided by the operating system.	MUST
8	IPv6 support	IPv6 (6lowpan) networking stack should be included in the operating system libraries to increase interoperability.	SHOULD
		Portability	
9	TelosB support	TelosB-compatible platform should be supported by WSN operating systems.	SHOULD
10	Rapid driver development	WSN operating systems must support implementation of additional sensor and other module drivers	MUST

...

Table 3 – continued

#	Rule	Description	Importance
11	Rapid platform definition	Porting to completely new platforms must be simple enough and operating systems should contain highly portable code.	MUST
12	802.15.4 support	Driver support for CC2420 radio or other 802.15.4-compatible radio communication chips should be provided by WSN operating systems.	SHOULD
13	AVR and MSP430 support	WSN operating systems must support Atmel AVR and Texas Instruments MSP430 MCU architectures.	MUST
	Task scheduling		
14	Low duty-cycle	WSN operating systems must set effective low-energy, low duty-cycle sampling as the first priority. High performance for sophisticated audio or other signal processing is secondary.	MUST
15	5 kernel + 6 user tasks	OS task scheduler must support up to 5 kernel services and up to 6 user level tasks.	MUST
16	Cooperative scheduling	Operating systems must provide cooperative tasks scheduling.	MUST
17	Preemptive scheduling	Operating systems should provide preemptive scheduling.	SHOULD
18	Event-based scheduling	Operating systems should provide event-based scheduling as an option.	SHOULD
	Services		
19	External storage	Interface for user data storage in external memory should be provided by WSN operating systems.	SHOULD
20	File system	Convenient file system interface should be provided by operating systems.	SHOULD
21	Time synchronization	Simple time synchronization should be provided by WSN operating systems.	SHOULD
	User support		
22	Base station example	WSN OS toolset must include an example base station application, which is easily extensible to user specific needs.	MUST
23	Popular sensor API	WSN operating system should provide common interface for temperature, light and acceleration sensor reading.	SHOULD
24	ADC API	ADC sampling interface must be provided by WSN operating systems.	MUST
25	Remote access	Remote data access and reprogramming of sensor nodes should be provided either by operating systems or other software abstractions.	SHOULD

In this paper the authors analyze how the proposed design rules can be used to solve common WSN problems. Mapping between design rules and problems is shown in Table 4. Problems and rules have M:N (many-to-many) relationship. I.e., each problem is addressed by several rules and each rule is addressing several problems. Design rule set is consistent, without contradictions.

Table 4. Addressing WSN problems by design rules (rules in rows, problems in columns)

	Design rules	#1 Chip reuse	#2 Field experts	#3 Hardware evolution	#4 Protocol variety	#5 WSN ≠ Internet	#6 Complex protocols	#7 Limited resources	#8 Experimentation	#9 QoS	#10 Energy	#11 Data caching	#12 Complex states	#13 Cooperation
Communication rules														
1	Sink-oriented					x	x	x						
2	Powered motes					x	x				x			
3	30 Byte payload					x								
4	11 hop routing		x		x	x		x			x			
5	CSMA MAC		x		x	x								
6	Custom protocol API								x	x	x			
7	Packet acknowledgment				x	x				x		x		
8	IPv6 support		x		x	x								
Portability rules														
9	TelosB support	x	x											
10	Rapid driver dev.	x		x										
11	Rapid platform def.	x		x										
12	802.15.4 support	x												
13	AVR & MSP430 support	x												
Task scheduling rules														
14	Low duty-cycle										x			
15	5 kernel + 6 user tasks		x										x	
16	Cooperative sched.		x								x		x	
17	Preemptive sched.		x										x	
18	Event-based sched.							x			x			
Service rules														
19	External storage		x									x		
20	File system		x									x		
21	Time Synchronization		x								x			x
User support rules														
22	Base-station example		x											
23	Popular sensor API		x	x										
24	ADC API	x	x	x										
25	Remote access		x						x					

The following sections discuss how design rules can be used to analyze existing WSN software solutions and plan their improvements both, for operating systems and applications.

7 Design rule impact on existing WSN deployments

In the deployment survey, described in previous sections, the authors identify trends that form basis for proposed design rules. However, none of the design rules is not satisfied by 100% of the deployments. Not all applications are optimal. Although different environments and constraints are encountered in deployments, some of the analyzed deployments can be optimized by adapting design rules. This section discusses the possible optimizations.

7.1 Rapid driver development and porting (design rules #10 and #11)

80% of deployments have involved custom driver development for either platform with specific components or porting the same application to another or completely custom platform (32.5%). Source code portability is important for WSNs as the platforms often evolve and development follows the prototyping model. Existing code modules as well as the operating system ideology and structure should support rapid and frequent changes. Unfortunately, the most popular operating system, TinyOS, follows ideology and contains source code that is hard to read and understand (distributed in various places, contains nesC specific constructs), and even harder to design during porting. While TinyOS might have high performance and resource efficiency, it should be improved dramatically in terms of usability. Although TinyOS us used here as an example (most popular OS choice among deployments), the portability and driver development rules are important for any WSN OS as these aspects impact many deployments.

7.2 Sink-oriented protocols and powered motes (design rules #1 and #2)

38 of 40 deployments (95%) needed a sink-oriented protocol and in 11 cases (27.5%) it was not provided by the operating system. Providing such protocol at OS or library level saves development time for users. Development of communication protocols is a complex task requiring thorough testing either in simulations or real pilot networks. Similarly, powered motes should be considered in these protocols (used in 82.5% of deployments, not provided by OS in 22.5%). By providing such protocols at OS level or libraries, users of 22.5% of deployments could have improved software development speed.

7.3 Custom protocol interface (design rule #6)

Interface for definition of custom MAC and routing protocols is essential part of operating system or middleware - this feature is required by 62.5% of deployments, and the requirement is satisfied only in 68% of cases when it is needed (42.5% of total deployments). This rule could decrease development time for WSN protocol researchers and encourage testing protocols on real platforms, as well as simulations, if the OS allows to compile application for simulated sensor nodes.

7.4 Cooperative scheduling (design rule #16)

In 62.5% cases a cooperative scheduling strategy is sufficient (preemption is not required). That implies that cooperative scheduling should be preferred as it is more efficient in different aspects, including efficient memory usage, less context switch time overhead and more appropriate task switch time selection. The design rule that suggests cooperative scheduling could improve 15% of deployments where cooperative scheduling is not provided by an operating system.

7.5 Sensor sampling interface (design rule #23)

In 65% of cases at least one of the most popular sensors (light, temperature, accelerometer) is used. Therefore operating system or middleware should provide a unified API for these sensor sampling. Some platform inspection functionality should be available telling the application what sensors are available. Unfortunately, such API is provided only in one of the cases where specific sensor extension board is used. By doing so one can assure that the same application can be run on different platforms.

7.6 Time synchronization (design rule #21)

In 35% of deployments some form of time synchronization is implemented in the application. Proper time synchronization requires complex algorithms, similarly to network protocols. Therefore it would be valuable to include basic time synchronization in the operating system. In 7.5% of deployments advanced and application specific time synchronization is used, which cannot be implemented at the OS level. However, in most of cases a generic time synchronization would suffice.

8 Impact on existing WSN operating systems

This section analyzes existing WSN OS conformance to proposed design rules and proposes potential improvements by identifying gaps. Summary of OS conformance is shown in Table 5.

8.1 TinyOS

As TinyOS complies to majority of rules, only the non-satisfied rules will be discussed in detail.

TinyOS disregards the following design rules : *design rule#10* (rapid driver development), *design rule#11* (rapid platform definition), *design rule#20* (file system), *design rule#21* (time synchronization), *design rule#23* (popular sensor API), and *design rule#25* (remote access). Although TinyOS is portable (wide range of supported platforms is a proof for it), code readability and simplicity is doubtful. The main reasons of TinyOS complexity are:

Table 5. Existing OS conformance to proposed design rules

#	Rule	TinyOS	Contiki	LiteOS	Mantis	MansOS	Arduino
	Communication						
1	Sink-oriented	+	+	+	+	+	
2	Powered motes	+	+			+	+
3	30 byte payload	+	+	+	+	+	+
4	11 hop routing	+	+	±	+	+	+
5	CSMA MAC	+	+			+	+
6	Custom protocol API	+	+			+	
7	Packet acknowledgment	+	+		+	+	+
8	IPv6 support	+	+				+
	Portability						
9	TelosB support	+	+		+	+	
10	Rapid driver development		+	+	+	+	+
11	Rapid platform definition		±		±	+	
12	802.15.4 support	+	+	+	+	+	+
13	AVR and MSP430 support	+	+	±	+	+	
	Task scheduling						
14	Low duty-cycle	+	+	+	+	+	
15	5 kernel + 6 user tasks	+	+	+	+	±	
16	Cooperative scheduling	+	+			+	
17	Preemptive scheduling	+	+	+	+	+	
18	Event-based scheduling	+	+			+	
	Services						
19	External storage	+	+	+	+	+	+
20	File system		+	+	+	+	+
21	Time synchronization		+			+	+
	User support						
22	Base station example	+		+	+	+	+
23	Popular sensor API			+	±	+	+
24	ADC API	+		+	+	+	+
25	Remote access		+	+	+	+	±

- The event-driven nature: while event handlers impose less overhead compared to sequential programming with blocking calls and polling, it is more complex for programmers to design and keep in mind the state machine for split-phase operation of the application.
- Modular component architecture: high degree of modularity and code reuse leads to program logic distribution into many components. Each new functionality may require modification in multiple locations, requiring deep knowledge of internal system structure.
- nesC language peculiarities: confusion of interfaces and components, component composition and nesting, specific requirements for variable definitions are examples of language aspects interfering with creativity of novice WSN programmers.

These limitations are in the system design level, and there is no quick fix available. The most convenient alternative is to implement middleware on top of TinyOS for simplified access to non-expert WSN programmers. TinyOS architecture is too specific and complex to introduce groundbreaking improvements for readability while maintaining backwards compatibility for existing applications.

Nevertheless, more than 100 groups around the world use TinyOS. It is also used by multiple commercial products (SOWNet technologies, 2014; Zolertia, 2014).

The rest of unsatisfied design rules regard to missing features that can be implemented as additions. And some of the functions are already implemented as external tools and middleware on-top of TinyOS. For example, third party external storage filesystem implementations do exist, such as TinyOS FAT16 support for SD cards (Goavec-Merou, 2010); Deluge can be used for remote reprogramming (Hui and Culler, 2004).

8.2 Contiki

Contiki does not provide platform independent API for temperature, light, and sound sensors (*design rule#23* (popular sensor API)) and ADC access (*design rule#24* (ADC API)). The reason is Contiki's mission - it is not dedicated specifically to sensor networks, rather to networked embedded device programming. Some of the platforms (such as Apple II) may not have sensors or ADC available, therefore the API is not explicitly enforced for all the platforms.

Portability to new platforms is partially effective (*design rule#11* (rapid platform definition)). MCU architecture code may be reused. However, large proportion of platform-specific code in Contiki may actually be reused on multiple platforms with appropriate restructuring.

Surprisingly, there is no base station application template included (*design rule#22* (base station example)). Contiki-collect is provided as an alternative - a complete and configurable sense-and-send network toolset for simple setup of simple sensor network applications.

To conclude, Contiki is one of the best WSN operating systems conforming with most of the proposed design rules.

8.3 LiteOS

LiteOS provides fully threaded programming with blocking calls, and no event callback handling (*design rule#18* (event-based scheduling)). No cooperative scheduler is provided (*design rule#16* (Cooperative scheduling)).

Networking stack is not included in the LiteOS distribution. However, multiple demo applications are usable as templates for user-specific networking protocol creation. Several routing protocols are implemented as user-level threads. The following communication rules are not satisfied: *design rule#2* (powered motes), *design rule#5* (CSMA MAC), *design rule#6* (custom protocol API), *design rule#7* (packet acknowledgement) and *design rule#8* (IPv6 support). LiteOS' applicability is very limited due to these inconsistencies.

LiteOS conforms to all user support rules, including interface for temperature, light and acceleration sensor sampling(*design rule#23*) and remote access (*design rule#25*). From service rules it lacks time synchronization support (*design rule#21*).

The source code is 8-bit AVR platform specific and significant changes are required to port LiteOS to other platforms with other microcontrollers. Chip driver development is relatively simple, as device drivers must implement only a predefined set of functions. However, new platform specification is unclear (*design rule#11* is not satisfied).

Compared to other WSN operating systems (TinyOS, Contiki and MansOS) LiteOS is a constrained OS with limited usability for field experts and other programmers who do not want to study or develop customized networking protocols.

8.4 Mantis

A TDMA-class MAC protocol supporting star network topology is included in the default configuration, without a CSMA MAC (*design rule#5* is not satisfied). No unified networking API is used, therefore users must design inter-layer interfaces on demand (*design rule#6* is not satisfied). In addition, the following rules are not satisfied by the existing networking implementation: *design rule#2* (powered motes), *design rule#7* (packet acknowledgement) and *design rule#8* (IPv6 support). Networking protocol stack of Mantis is not thoroughly developed, and development of the OS itself has stopped in recent years.

Platform- and chip-level code is mixed, there are no TelosB or MicaZ platforms, rather MSP430 and AVR code, which is MCU or architecture specific. Separation of MCU architectures, specific chips and platforms would improve portability (*design rule#11*).

Only preemptive thread scheduling is supported by the OS which, similarly to LiteOS, limits its efficiency for constrained application class. No cooperative scheduling (*design rule#16*) or event-based scheduling (*design rule#18*) is provided.

Mantis is rich in supported API, services and examples, yet no time synchronization is provided (*design rule#21*).

Mantis has a promising software base that would be extensible for a rich WSN OS. Unfortunately, it's development activity has stopped.

8.5 MansOS

MansOS is a wireless sensor network operating system developed with simplicity of use and portability in mind. The authors have also participated in MansOS development since it's start in 2008. MansOS is still actively developed now. More MansOS details in (Elsts et al., 2012).

During it's development, ideology and core components of MansOS have evolved in multiple iterations. Therefore the existing version conforms to most of the design rules proposed in this paper. There are some exceptions and space for improvement that will be discussed here.

The authors of this paper introduced cooperative task scheduler (described in (Strazdins, 2014)) to MansOS as a result of design rule development (*design rule#16*). Previously MansOS was supporting two scheduling techniques: direct event handling and preemptive scheduler. Based on the findings in deployment survey development team decided that cooperative scheduler is an important part of WSN OS. As a result of multiple alternative evaluation, one of the authors decided to integrate *ProtoThreads* scheduler from Contiki OS (Dunkels et al., 2006) - it has been proved to work stable already in Contiki, therefore there was no need to *reinvent the wheel*. It is an example of how design rules improved WSN OS in practice - by substantiating importance of a particular feature that was not implemented previously.

Two improvements are required in MansOS to reach full conformance to proposed design rules. First, IPv6 support is required (*design rule#8*). While third-party IPv6 libraries can be used (Dunkels, 2003), such addition would interfere with the existing networking protocol infrastructure. IPv6 should be fully integrated as an optional component in the common protocol stack. Second, the preemptive scheduler is limited to only one kernel thread at the moment while *design rule#15* states that 5 kernel tasks should be supported. This can be fixed by implementing a multi-threaded kernel, although it might require some re-design of the whole OS. Otherwise, two problems may arise. First, the tasks running in kernel context have equal priority, it is not possible to assign higher priority to any of the tasks. Second, the kernel tasks are implicit without possibility to create libraries of additional kernel tasks, that can be loaded and unloaded as necessary.

In summary, MansOS demonstrates how design rules are applied both during OS development phase and also in evaluation to detect potential problems and design necessary improvements.

8.6 Arduino

This section discusses how Arduino conforms to proposed WSN software development design rules and how Arduino can be modified to become a fully-functional WSN OS.

Core Arduino OS provides only basic MCU driver and a base for extensions. However, Arduino is a community-based project without strict borders of OS and third-party software. It is a set of solutions and libraries that are combined during custom solution development. Therefore here we examine the opportunities of core Arduino together with libraries and extensions that are widely accepted. As the Arduino solution is based

on engineer and enthusiast community (instead of WSN researchers) most of the references in this section point to web sites instead of scientific articles. Nevertheless, these sites do not have hypothesis and statements that have to be proved. Instead they contain source code libraries and examples that can be simply verified empirically. Therefore these sources are sufficiently reliable for this particular section.

Arduino core contradicts to all network rules as there is only a microcontroller on the base board and USB is the only communication with a PC. However Zigbee/XBee-802.15.4 modules are available providing networking options. Zigbee has a built-in mesh capability. Networking rules are described here, using XBee Series 2 modules with Zigbee stack (Arduino SA, 2014b). In addition, new versions of Arduino boards with built-in communication do appear, such as Arduino Yun (SA, 2014) and Flutter (Flutter Wireless, 2013).

The Zigbee networking protocol stack implements an 802.15.4-compatible (*design rule#12*) mesh network topology, without sink-oriented data flow architecture (*design rule#1* is not satisfied). Powered motes are considered in Zigbee, called *coordinator* (Faludi, 2010) (*design rule#2* is satisfied (powered mote support)). Zigbee includes CMSA-based MAC protocol and Ad hoc On-demand Distance Vector (AODV) routing (Perkins and Royer, 1999) with reliable packet delivery (*design rules #4, #5 and #7* are satisfied). IPv6 library from Contiki has been ported to Arduino (Baptiste Gaultier, 2013) (*design rule#8* satisfied). However, the MAC and routing protocols are predefined and cannot be customized by the user (*design rule#6* not satisfied).

Arduino platform has space for improvement regarding portability. It is designed only for AVR-based microcontrollers, TelosB platform (*design rule#9*) and MSP430-family MCUs (*design rule#13* (AVR and MSP430 support)) are not supported. Arduino is not designed for other architectures, therefore its software is not ready for porting: *design rule#11* (rapid platform definition) is not satisfied. Nevertheless, driver development (*design rule#10*) is facilitated with wide range of existing sensor and other chip drivers, libraries and examples.

Task scheduling in Arduino is not optimal for WSNs. It uses a simple single-thread polling approach. Protothread library (from Contiki) can be used on Arduino (Dunkels et al., 2006), but it there is no ready-to-use Protothread port or tutorial available. All task scheduling design rules are not satisfied. An advanced task scheduler is needed to adapt Arduino for WSN needs.

Arduino supports wide range of services and interfaces using community-contributed libraries, all service and user support design rules are satisfied. Examples include external memory and FAT file system (Greiman, 2013), time synchronization (Arduino SA, 2011), ADC and wide range of sensors drivers (Arduino SA, 2014c). Remote reprogramming is possible by external tools requiring custom bootloader (CodeBender team, 2014).

To summarize, Arduino needs addition of task scheduler, portability to low-power hardware platforms and more flexibility for networking protocols.

8.7 Summary

The evaluation shows that popular WSN operating systems conform to majority of proposed design rules. However, each OS has some specific aspects that can be improved.

This paper can serve as a reference for OS developers substantiating importance of particular design rules.

9 Use-case study: a wearable sensor network

This case study describes a research project on tactile ship bridge alarm system development, performed jointly by Maritime Human Factors Laboratory at Aalesund University College and Rolls Royce Marine, Norway. One of the authors has participated in the project as one of the main software and hardware designers and developers. The author has implemented a device prototype. In the case study the author analyzes possible improvements of the tactile alarm system that can be introduced by implementing the proposed design rules.

The system consists of two parts (Figure 3). One part is implemented as an add-on for the ship bridge system. It takes information about person location and actual alarms, and generates tactile alarm signals to be sent to persons. The second part is a system worn on the operators. It receives commands wirelessly and generates tactile cue patterns for the actuators mounted on the person. After survey of different device placement options, the author chose tactile belt as the most appropriate form for first prototype. Its main advantages: close contact with the person, naturalness, ability to disseminate cues 24 hours daily, as well as ability to give directional cues.

Although the main focus of the system is actuation, this system is a wireless sensor network. In further revisions the system would contain sensor modality, such as position and pose estimation. For deployment at least two belts are required for maritime operators and optional belts for other crewmembers. Continuous connectivity would require a wireless base station and router infrastructure that is able to provide two-way communication with the mobile, wearable.

A tactile belt prototype is show in Figure 4.

The prototype consists of:

- Bluetooth radio module acting as a bridge between the belts and external alarm system. Bluetooth RFCOMM profile is used emulating virtual serial port.
- Four vibrator motors generating tactile cues placed uniformly along the belt.
- Arduino LilyPad MCU translating received commands into motor commands.
- Lithium Polymer battery as a power source.
- Power regulator module, translating unstable 3.7V battery voltage into stable 5.0V voltage used by the system. The software functions in server/client mode where ships alarm system (a custom Java application used as a stub in experiments) operates as a server that sends commands to clients tactile belts.

The following design rules are not satisfied in the prototype implementation:

- Multi-hop network support. Bluetooth RFCOMM is used, without routing.
- Custom MAC and routing protocol development interface. Provided neither by Arduino, nor Bluetooth.
- IPv6 support. Bluetooth uses local addressing and IPv6 is not supported.
- TelosB platform is not supported. Arduino does not conform TelosB specification neither by chip characteristics, nor energy efficiency.

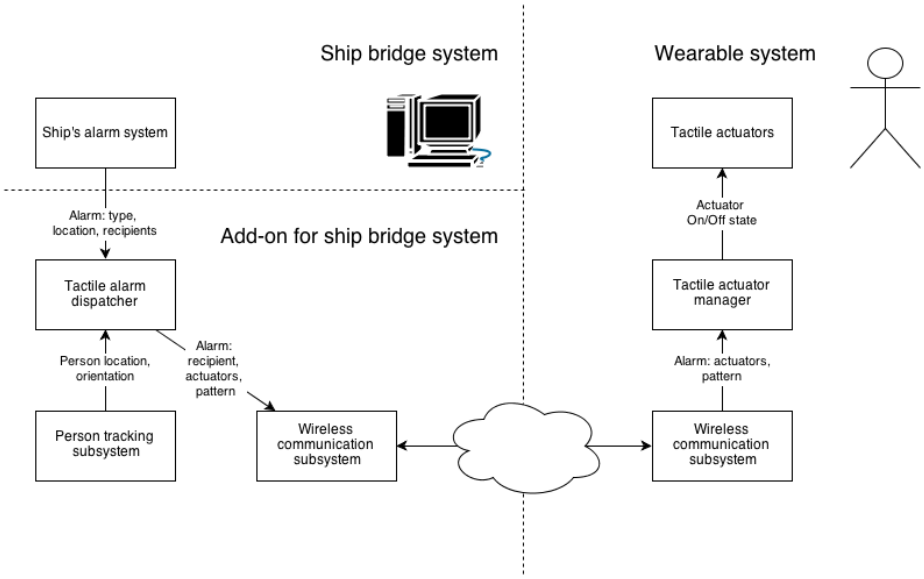

Fig. 3. Tactile ship bridge alarm system architecture - wearable sensor and actuator device communicating with ship bridge automation system

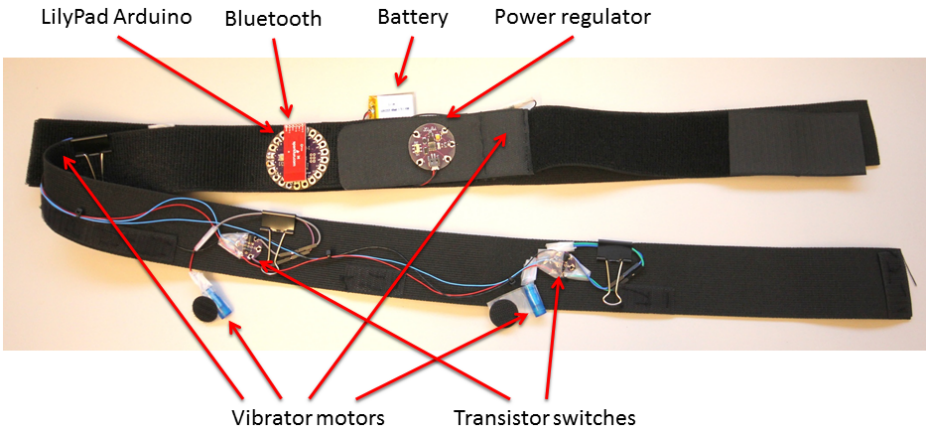

Fig. 4. Tactile belt prototype - Vibrator motors, microcontroller, battery power source and wireless communication mounted on a stretchable material

- Portability is limited and no MSP430 support is available.
- No task scheduling techniques are used, no multitasking support.

The following problems have been identified in the prototype:

- Short network lifetime. Real deployments would require at least 7 days of autonomous operation, not provided by the prototype.
- No multi-hop network support. One-hop network is usable in a single room. It limits event forwarding to external persons, including bridge watch.
- No multitasking support. All processes (sensor sampling, packet reception, packet transmission and motor handling) can be implemented in a single thread, however, that would be complex and contradict with systems logic.

The authors analyze improvements that are suggested by the proposed design rules in the remainder of this section.

9.1 Network lifetime extension

There are two modes of expected system operation:

- During intense operations, where tactile device might inform the operators of critical information, low latency is important (in millisecond range). Therefore 100% radio duty-cycle is expected here.
- During other time (actually, most of time) crew members are in idle mode, when nothing significant is happening. They must be warned in case of alarm, yet the acceptable latency is much higher (might be several seconds).

Most energy is spent in radio listening mode. Customized MAC protocols (*design rule#6*) that allow changing radio duty cycle can help to reduce energy consumption significantly. For example, if the radio transmission is activated every 5 seconds for a 250ms period (it takes around 100ms to send a 46-byte packet (Amiri, 2010), 250ms is enough for two-way communication), it results in a 20% duty cycle.

The current Bluetooth module does not allow control of MAC protocols. Therefore more efficient radio module must be selected. In addition, the Arduino board with AVR ATMega328 microcontroller is also not the best option in terms of energy efficiency it consumes around 25mA in active mode, and additional 25mA for Bluetooth radio, the total consumption of the platform is more than 50mA or less than 8 hours of operation from a 400mAh battery.

Vibrator motor energy consumption cannot be accurately predicted without a particular scenario. The motors will be active in a very tiny fraction of time, the duty cycle will be very low. Let us examine an example scenario where 4 motors are used, each of them consumes 50mA of energy in active mode. The operators are active 8 hours daily performing operations where alarms may be raised once every 5 minutes and the motors are active for 1 second on every alarm. That means a 96 seconds of active motors during the 8 hour operation or 5.33mAh of total energy consumed. During the inactive period of the day the probability of an alarm is low, let us approximate it to one alarm every day. However, the motors will be active longer on each alarm, let us define the activity period 60 seconds in this case the person must react and turn the alarm off in one

minutes time. That leads to 60 seconds of motors in active mode or 3.33mAh of energy consumed. Taken together, less than 9mAh of energy is consumed daily for the motor operation or less than 0.375mAh of average consumption. Although this example uses multiple assumed constants, it shows that the motor energy consumption in a realistic scenario is insignificant, compared to consumption of the rest of the system.

Selection of an energy-efficient wearable sensor-actuator node increases the lifetime dramatically. Let us take a TelosB-compatible platform with MSP430F1611[3] microcontroller and CC2420 radio, such as TMote Sky, as an example. The whole platform consumes 20-23mA during active radio transmission or reception. With a 20% duty-cycle that would result in less than 5mA average consumption. It is tenfold increase in energy efficiency compared to existing implementation. To conclude, a solution that supports custom MAC protocols (*design rule#6*), TelosB-compatible platform (*design rule#9*) and low duty-cycle (*design rule#14*), would lead to significant lifetime extension.

In addition, it is important to be able to experiment with multiple different platforms and select the best alternatives based on empirical evidence. Rapid porting and driver development (*design rules #10 and 11*) are important in that matter. To comply with these rules, Arduino software should be replaced with a solution more appropriate for porting to new platforms and providing wider set of supported platforms. Contiki OS would be a good candidate: the solution can be incrementally ported to Contiki OS keeping the same initial hardware and than changing hardware component-by-component as necessary.

9.2 Multi-hop communication

To implement a deployable system, alarm dissemination is required also outside the ship bridge room, and a 24-hour stable operation is required. Multi-hop communication (*design rule#4*) between the alarm generation system and tactile wearable devices is essential part of this requirement. The solution can be implemented in multiple different ways: either the conventional ship automation systems network (TCP/IP or other) is used to create a backbone network and connect tactile devices using gateway nodes attached to each backbone network router, or a mesh network of wearable devices and corresponding sensor network routers (802.15.4) can be installed on the ship, connected to the automation systems network using a single (or multiple redundant) gateway nodes.

9.3 Multitasking support

There are multiple logical tasks running concurrently on the wearable device: motor control, data reception, data transmission and sensor sampling (no sensors attached at the moment, but could be required in future deployments). Support of multi-tasking by providing API for separate thread creation (*design rule#15*) is necessary due to different aspects. First, it is correct to separate and encapsulate threads with different responsibilities and resources. It is logically more correct and makes the code easier to maintain and expand. Second, correct multi-tasking can improve the efficiency of the application in terms of time-sharing threads wait when they have no operation to perform and start

[3] http://www.ti.com/product/msp430f1611

running whenever the expected event has occurred. Fully accurate multitasking is not achievable on a single-processor microcontroller, yet the idle-time can be minimized.

Selection of scheduling techniques depends on task characteristics. If some of them are time-critical (MAC protocol) while others may be time-intensive (data processing), preemptive scheduling is required (*design rule#17*). If there is no intensive data processing, only command execution and sensor data reports, cooperative scheduler (*design rule#16*) is sufficient and will have less overhead on average. As the whole system is event-driven, event-based scheduling with configuration on callback function (*design rule#18*) would be very efficient in terms of system performance, yet it might be more difficult for the programmers if the system grows more complicated during its evolution.

9.4 Use case summary

This section describes a wireless sensor network use case where one of the authors created a prototype implementation of wearable wireless device. The authors identified several problems, including short network lifetime, limited communication abilities and problematic source code design and maintenance. Then we showed that following several of the WSN design rules proposed in this paper can make significant improvements and can solve the identified problems. To conclude, this use case showed that the design rules are applicable for WSN quality assurance as a diagnostics checklist and also solution guide.

10 Conclusion

Central thesis of this work states that wireless sensor network software development lack a unified methodology that provides network interoperability, higher source code reusability and portability to other platforms. The goal of this paper is to provide unified WSN software development methodology in the form of a design rule set. To reach the goal the authors analyze typical WSN deployments described in scientific literature and infer common problems and requirements. Design rules are proposed on this deployment survey. The applicability of these rules is evaluated in different aspects - the authors show the impact of the rules for existing deployment and also operating system evaluation. Considering wide variety of sensor network applications, it is not possible to define design rules as theorems that can be formally proved. However, even in the existing guideline form the design rules represent valuable knowledge for WSN researchers and users. To the best of the authors' knowledge, this work proposes the most comprehensive and formalized set of design rules for WSN software development.

Future work includes design rule evaluation and adaption for particular WSN subsets, such as body sensor networks or smart homes; as well as rule testing in external projects by researchers other than the authors.

Acknowledgements

This work has been supported by the European Social Fund within the project "Support for Doctoral Studies at University of Latvia" and Latvian National Research Program

Development of innovative multi-functional material, signal processing and information technologies for competitive and research intensive products. Thank you for productive teamwork to our former EDI colleagues: Artis Mednis, Atis Elsts, Reinholds Zviedris, Georgijs Kanonirs, Krisjanis Nesenbergs and Modris Greitans. Thank you also to Guntis Arnicans who improved the structure and focus of the work significantly by constructive discussion.

References

Amiri, M. (2010). Measurements of energy consumption and execution time of different operations on Tmote Sky sensor motes. Master's thesis, Masaryk University.

Arduino SA (2011). Arduino Time Library. http://playground.arduino.cc/Code/time.

Arduino SA (2014a). Arduino. http://arduino.cc/.

Arduino SA (2014b). Arduino Wireless Shield with XBee Series 2 radios. http://arduino.cc/en/Guide/ArduinoWirelessShieldS2.

Arduino SA (2014c). Interfacing with Hardware: Input. http://playground.arduino.cc/Main/InterfacingWithHardware.

Arora, A., Dutta, P., Bapat, S., Kulathumani, V., Zhang, H., Naik, V., Mittal, V., Cao, H., Demirbas, M., Gouda, M., Choi, Y., Herman, T., Kulkarni, S., Arumugam, U., Nesterenko, M., Vora, A., and Miyashita, M. (2004). A line in the sand: a wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605 – 634. Military Communications Systems and Technologies.

Aylward, R. and Paradiso, J. A. (2007). A compact, high-speed, wearable sensor network for biomotion capture and interactive media. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 380–389, New York, NY, USA. ACM.

Baptiste Gaultier (2013). Arduino uipv6 stack. https://github.com/telecombretagne/Arduino-IPv6Stack/wiki.

Barrenetxea, G., Ingelrest, F., Schaefer, G., and Vetterli, M. (2008). The hitchhiker's guide to successful wireless sensor network deployments. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 43–56, New York, NY, USA. ACM.

Beutel, J. (2006). Metrics for Sensor Network Platforms. In *Proc. ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'06)*, page 5.

Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., Shucker, B., Gruenwald, C., Torgerson, A., and Han, R. (2005). Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Networks and Applications*, 10(4):563–579.

Burke, J., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., and Srivastava, M. (2006). Participatory Sensing. In *Proc. of World Sensor Web Workshop (WSW'06), collocated with SenSys'06*, pages 1–5.

Butler, Z., Corke, P., Peterson, R., and Rus, D. (2004). Virtual fences for controlling cows. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4429–4436.

Cao, Q., Abdelzaher, T., Stankovic, J., and He, T. (2008). The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 233–244, Washington, DC, USA. IEEE Computer Society.

Ceriotti, M., Corrà, M., D'Orazio, L., Doriguzzi, R., Facchin, D., Guna, S., Jesi, G., Cigno, R., Mottola, L., Murphy, A., et al. (2011). Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels. In *Proceedings of the 10th ACM/IEEE*

International Conference on Information Processing in Sensor Networks (IPSN/SPOTS), pages 187–198.

Ceriotti, M., Mottola, L., Picco, G. P., Murphy, A. L., Guna, S., Corra, M., Pozzi, M., Zonta, D., and Zanon, P. (2009). Monitoring heritage buildings with wireless sensor networks: The torre aquila deployment. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 277–288, Washington, DC, USA. IEEE Computer Society.

Chebrolu, K., Raman, B., Mishra, N., Valiveti, P., and Kumar, R. (2008). Brimon: a sensor network system for railway bridge monitoring. In *Proceedings of the 6th international conference on Mobile systems, applications, and services (MobiSys)*, pages 2–14.

CodeBender team (2014). codebender.

Detweiler, C., Doniec, M., Jiang, M., Schwager, M., Chen, R., and Rus, D. (2010). Adaptive decentralized control of underwater sensor networks for modeling underwater phenomena. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 253–266, New York, NY, USA. ACM.

Dunkels, A. (2003). Full tcp/ip for 8-bit architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services (MobiSys'03).*, pages 85–98. ACM.

Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. of Annual IEEE Conference on Local Computer Networks*, pages 455–462, Los Alamitos, CA, USA. IEEE Computer Society.

Dunkels, A., Schmidt, O., Voigt, T., and Ali, M. (2006). Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. In *Proc. of SenSys'06*, pages 29–42.

Dyo, V., Ellwood, S. A., Macdonald, D. W., Markham, A., Mascolo, C., Pásztor, B., Scellato, S., Trigoni, N., Wohlers, R., and Yousef, K. (2010). Evolution and sustainability of a wildlife monitoring sensor network. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 127–140, New York, NY, USA. ACM.

Eisenman, S. B., Miluzzo, E., Lane, N. D., Peterson, R. A., Ahn, G.-S., and Campbell, A. T. (2010). Bikenet: A mobile sensing system for cyclist experience mapping. *ACM Trans. Sen. Netw.*, 6:6:1–6:39.

Elsts, A., Strazdins, G., Vihrov, A., and Selavo, L. (2012). Design and implementation of mansos: a wireless sensor network operating system. In *Scientific Papers*. University of Latvia.

Faludi, R. (2010). *Building wireless sensor networks: with ZigBee, XBee, Arduino, and Processing*. O'reilly.

Finne, N., Eriksson, J., Dunkels, A., and Voigt, T. (2008). Experiences from two sensor network deployments: self-monitoring and self-configuration keys to success. In *Proceedings of the 6th international conference on Wired/wireless internet communications*, WWIC'08, pages 189–200, Berlin, Heidelberg. Springer-Verlag.

Flutter Wireless (2013). Wireless ARM development board with over 1 km range. http://www.flutterwireless.com/.

Franceschinis, M., Gioanola, L., Messere, M., Tomasi, R., Spirito, M., and Civera, P. (2009). Wireless sensor networks for intelligent transportation systems. In *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, pages 1 –5.

Gao, T., Massey, T., Selavo, L., Crawford, D., rong Chen, B., Lorincz, K., Shnayder, V., Hauenstein, L., Dabiri, F., Jeng, J., Chanmugam, A., White, D., Sarrafzadeh, M., and Welsh, M. (2007). The advanced health and disaster aid network: A light-weight wireless medical system for triage. *Biomedical Circuits and Systems, IEEE Transactions on*, 1(3):203–216.

Goavec-Merou, G. (2010). SDCard and FAT16 file system implementation for TinyOS http://www.trabucayre.com/page-tinyos.html. http://www.trabucayre.com/page-tinyos.html.

Greiman, B. (2013). sdfatlib: A FAT16/FAT32 Arduino library for SD/SDHC cards. https://code.google.com/p/sdfatlib/.

Hac, A. (2003). *Wireless sensor Network Designs*. John Wiley and Sons, Ltd.

Handziski, V., Kopke, A., Karl, H., and Wolisz, A. (2003). A common wireless sensor network architecture? Technical report, Telecommunications Networks Group, Technische Universitat Berlin.

Hartung, C., Han, R., Seielstad, C., and Holbrook, S. (2006). Firewxnet: a multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, MobiSys '06, pages 28–41, New York, NY, USA. ACM.

He, T., Krishnamurthy, S., Stankovic, J. A., Abdelzaher, T., Luo, L., Stoleru, R., Yan, T., Gu, L., Hui, J., and Krogh, B. (2004). Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 270–283, New York, NY, USA. ACM.

Hill, J., Horton, M., Kling, R., and Krishnamurthy, L. (2004). The platforms enabling wireless sensor networks. *Commun. ACM*, 47:41–46.

Hill, J. L. (2003). *System architecture for wireless sensor networks*. PhD thesis, University of California.

Ho, L., Moh, M., Walker, Z., Hamada, T., and Su, C.-F. (2005). A prototype on rfid and sensor networks for elder healthcare: progress report. In *Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, E-WIND '05, pages 70–75, New York, NY, USA. ACM.

Huang, R., Song, W.-Z., Xu, M., Peterson, N., Shirazi, B., and LaHusen, R. (2012). Real-world sensor network for long-term volcano monitoring: Design and findings. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):321–329.

Hui, J. W. and Culler, D. (2004). The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 81–94, New York, NY, USA. ACM.

IEEE (2014). IEEE 802.15 WPAN Task Group 4 (TG4). http://www.ieee802.org/15/pub/TG4.html.

Ince, N. F., Min, C.-H., Tewfik, A., and Vanderpool, D. (2008). Detection of early morning daily activities with static home and wearable wireless sensors. *EURASIP J. Adv. Signal Process*, 2008.

Jarochowski, B., Shin, S., Ryu, D., and Kim, H. (2007). Ubiquitous rehabilitation center: An implementation of a wireless sensor network based rehabilitation management system. In *Convergence Information Technology, 2007. International Conference on*, pages 2349 –2358.

Jiang, X., Dawson-Haggerty, S., Dutta, P., and Culler, D. (2009). Design and implementation of a high-fidelity ac metering network. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 253–264, Washington, DC, USA. IEEE Computer Society.

Kansal, A., Hsu, J., Zahedi, S., and Srivastava, M. B. (2007). Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems, Special Section LCTES'05*, 6.

Krishnamurthy, L., Adler, R., Buonadonna, P., Chhabra, J., Flanigan, M., Kushalnagar, N., Nachman, L., and Yarvis, M. (2005). Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 64–75, New York, NY, USA. ACM.

Lai, T.-t. T., Chen, Y.-h. T., Huang, P., and Chu, H.-h. (2010). Pipeprobe: a mobile sensor droplet for mapping hidden pipeline. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 113–126, New York, NY, USA. ACM.

Langendoen, K., Baggio, A., and Visser, O. (2006). Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 1–8. IEEE.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., et al. (2005). Tinyos: An operating system for sensor networks. *Ambient intelligence*, 35.

Li, M. and Liu, Y. (2009). Underground coal mine monitoring with wireless sensor networks. *ACM Trans. Sen. Netw.*, 5:10:1–10:29.

Lifton, J., Feldmeier, M., Ono, Y., Lewis, C., and Paradiso, J. A. (2007). A platform for ubiquitous sensor deployment in occupational and domestic environments. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 119–127, New York, NY, USA. ACM.

Liu, L. and Ma, H. (2006). Wireless sensor network based mobile pet game. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, New York, NY, USA. ACM.

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. (2002). Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, New York, NY, USA. ACM.

Malinowski, M., Moskwa, M., Feldmeier, M., Laibowitz, M., and Paradiso, J. A. (2007). Cargonet: a low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 145–159, New York, NY, USA. ACM.

Mednis, A., Strazdins, G., Liepins, M., Gordjusins, A., and Selavo, L. (2010). RoadMic: Road Surface Monitoring Using Vehicular Sensor Networks with Microphones. In *Proc. of Networked Digital Technologies, Part II: Second International Conference, NDT 2010*, pages 417–429. Springer-Verlag GmbH.

Merrill, W., Newberg, F., Sohrabi, K., Kaiser, W., and Pottie, G. (2003). Collaborative Networking Requirements for Unattended Ground Sensor Systems. In *Proc. of IEEE Aerospace Conference*.

Mhatre, V. and Rosenberg, C. (2004). Design guidelines for wireless sensor networks: communication, clustering and aggregation. *Ad Hoc Networks*, 2(1):45–63.

Mottola, L. and Picco, G. P. (2011). Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Computing Surveys*, 43:19:1–19:51.

Mount, S., Gaura, E., Newman, R. M., Beresford, A. R., Dolan, S. R., and Allen, M. (2005). Trove: a physical game running on an ad-hoc wireless sensor network. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, sOc-EUSAI '05, pages 235–239, New York, NY, USA. ACM.

Olariu, S. and Stojmenovic, I. (2006). Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12.

Oppermann, F. J. and Peter, S. (2010). Inferring technical constraints of a wireless sensor network application from end-user requirements. In *Mobile Ad-hoc and Sensor Networks (MSN), 2010 Sixth International Conference on*, pages 169–175. IEEE.

Perkins, C. E. and Royer, E. M. (1999). Ad-hoc on-demand distance vector routing. In *Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99).*, pages 90–100. IEEE.

Romer, K. and Mattern, F. (2004). The design space of wireless sensor networks. *Wireless Communications, IEEE*, 11(6):54–61.

SA, A. (2014). Arduino Yún.

Santos, V., Bartolomeu, P., Fonseca, J., and Mota, A. (2007). B-live - a home automation system for disabled and elderly people. In *Industrial Embedded Systems, 2007. SIES '07. International Symposium on*, pages 333 –336.

Selavo, L., Wood, A., Cao, Q., Sookoor, T., Liu, H., Srinivasan, A., Wu, Y., Kang, W., Stankovic, J., Young, D., and Porter, J. (2007). Luster: wireless sensor network for environmental research. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 103–116, New York, NY, USA. ACM.

Sharp, C., Schaffert, S., Woo, A., Sastry, N., Karlof, C., Sastry, S., and Culler, D. (2005). Design and implementation of a sensor network system for vehicle tracking and autonomous interception. In *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 93 – 107.

Simon, G., Maróti, M., Lédeczi, A., Balogh, G., Kusy, B., Nádas, A., Pap, G., Sallai, J., and Frampton, K. (2004). Sensor network-based countersniper system. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 1–12, New York, NY, USA. ACM.

Song, H., Zhu, S., and Cao, G. (2008). Svats: A sensor-network-based vehicle anti-theft system. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 2128 –2136.

SOWNet technologies (2014). G-Node. http://www.sownet.nl/index.php/en/products/gnode.

Stojmenovic, I., Nayak, A., and Kuruvila, J. (2005). Design guidelines for routing protocols in ad hoc and sensor networks with a realistic physical layer. *Communications Magazine, IEEE*, 43(3):101–106.

Strazdins, G. (2014). *Wireless Sensor Network Software Design Rules*. PhD thesis, University of Latvia.

Strazdins, G., Elsts, A., and Selavo, L. (2010). MansOS: Easy to Use, Portable and Resource Efficient Operating System For Networked Embedded Devices. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 427–428, New York, NY, USA. ACM.

Suh, C., Ko, Y.-B., Lee, C.-H., and Kim, H.-J. (2006). The Design and Implementation of Smart Sensor-based Home Networks. In *Proc. of the International Symposium on Ubiquitous Computing Systems (UCS'06)*, page 10.

Thorstensen, B., Syversen, T., Bjørnvold, T.-A., and Walseth, T. (2004). Electronic shepherd - a low-cost, low-bandwidth, wireless network system. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 245–255, New York, NY, USA. ACM.

Tilak, S., Abu-Ghazaleh, N. B., and Heinzelman, W. (2002). A taxonomy of wireless microsensor network models. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(2):28–36.

Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., and Welsh, M. (2006). Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 381–396, Berkeley, CA, USA. USENIX Association.

Wilson, J., Bhargava, V., Redfern, A., and Wright, P. (2007). A wireless sensor network and incident command interface for urban firefighting. In *Mobile and Ubiquitous Systems: Networking Services, 2007. MobiQuitous 2007. Fourth Annual International Conference on*, pages 1 –7.

Wittenburg, G., Terfloth, K., Villafuerte, F. L., Naumowicz, T., Ritter, H., and Schiller, J. (2007). Fence monitoring: experimental evaluation of a use case for wireless sensor networks. In

Proceedings of the 4th European conference on Wireless sensor networks, EWSN'07, pages 163–178, Berlin, Heidelberg. Springer-Verlag.

Wood, A., Virone, G., Doan, T., Cao, Q., Selavo, L., Wu, Y., Fang, L., He, Z., Lin, S., and Stankovic, J. (2006). Alarm-net: Wireless sensor networks for assisted-living and residential monitoring. Technical report, University of Virginia Computer Science Department.

Zhang, P., Sadler, C. M., Lyon, S. A., and Martonosi, M. (2004). Hardware design experiences in zebranet. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 227–238, New York, NY, USA. ACM.

Zolertia (2014). Z1 Platform. http://www.zolertia.com/ti.

Received June 27, 2014, accepted June 30, 2014.