Software Engineering and Wireless Sensor Networks: Happy Marriage or Consensual Divorce?

Gian Pietro Picco
Department of Information Engineering and Computer Science (DISI)
University of Trento, Italy
gianpietro.picco@unitn.it

ABSTRACT

The development of wireless sensor networks (WSNs) software today is tackled by a code-and-fix process that relies solely on the primitive constructs provided by the operating system and the skills of developers. For WSNs to emerge from research labs and make a true impact on society at large, we need methodologies, techniques, and abstractions that improve the development process, foster the designer's confidence about the WSN behavior, and whose effectiveness is demonstrated in the real world.

How do we achieve these goals? The aforementioned challenges are germane to the techniques and expertise matured by software engineering (SE). Unfortunately, the WSN and SE research communities have been mostly impermeable to each other. In this paper we elaborate on this state of affairs, by arguing that a principled approach to development is inevitable as WSNs become more and more pervasive, and by identifying and discussing specific areas where a synergy between the SE and WSN communities could provide immediate, much-needed results.[1]

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-based Systems**]: Real-time and embedded systems; D.2 [**Software Engineering**]

General Terms

Design, Experimentation, Performance, Reliability

Keywords

Wireless sensor networks, software engineering

1. INTRODUCTION & MOTIVATION

Wireless sensor networks (WSNs) are distributed systems composed of tiny, resource-scarce, often battery-powered devices that cooperate to perform distributed monitoring and control over a self-organized network topology. Due to their miniaturization, autonomy, and flexibility, WSNs are a key element in many pervasive computing scenarios.

Software plays a fundamental role in WSNs. Unlike conventional distributed systems, where the application layer is built atop a largely immutable, application-agnostic network stack, in WSNs the latter is *i)* entirely realized in software, therefore malleable, and *ii)* expected to be tailored to the needs of the application.

However, WSN software is also quite complex. The application and the network protocols must often be co-designed, to satisfy the end-user needs. To complicate matters, the latter include not only functional requirements, but also stringent non-functional requirements about lifetime, reliability, and data yield. Meeting these requirements with resource-limited devices is arguably the defining challenge of WSNs.

WSN Development Today: Code-and-Fix Reloaded. Given that software plays a fundamental role in WSNs, and that WSNs have been around for a decade, one could be led to think that a satisfactory toolset of methodologies and techniques exists to properly support developers in meeting the application requirements.

Unfortunately, reality is much less encouraging. Today, the challenges above are tackled by a *code-and-fix* development process that relies solely on the rather primitive constructs provided by the operating system—if an OS is actually provided, that is—and on the developer's skills. One could argue that just like WSN hardware is about as powerful as computers from the '70s, the software development process remained stuck at about the same time. This situation hampers reuse and maintenance, because software modules tend to be highly coupled and very much geared towards the needs of the application at hand. Moreover, it constitutes a barrier towards a direct use of this technology by domain experts.

Why is that? There are at least a couple of arguments that can be brought to the table to try explaining the situation. The first one is what can be called "community impedance". In a private, informal conversation about WSN at large, a fellow researcher once commented to us, somewhat bitterly, *"The WSN field has been hijacked by the networking guys"*. This comment stigmatizes the large fraction of WSN research that has been devoted to routing and MAC protocols, therefore neglecting other aspects, including software development. Nevertheless, while the networking community is still a major player in the WSN arena, it is also true that today several other communities have joined the effort, and to some extent redefined the research agenda. Interestingly, the software engineering one is not among them: we come back to this later.

Another reason has to do with the role of deployments in current WSN research. The number of real-world deployments reported in the literature has increased significantly in the last few years, effec-

[1]This position paper is based on a keynote given by the author, for which a 1-page abstract appeared in [9].

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

Figure 1: Needs of wireless sensor networks and challenges for software engineering.

tively demonstrating the depth and breadth of the potential of this technology. However, too often these deployments are approached as short-lived experiments yielding just enough results to demonstrate the feasibility of a technique, rather than as reliable components of long-lived, operational systems. In this mindset, driven more by publication goals more than by real-world needs, building ad hoc solutions is perfectly acceptable. However, this is clearly not acceptable in the long run. This situation clearly reduces the credibility of the field when industrial-strength large investments are considered outside of a research setting.

Needs and Challenges. For WSNs to emerge from research labs and become useful in the real world, there is an obvious need for methodologies, techniques, and abstractions that (Figure 1):

- improve the development process, e.g., by simplifying programming, fostering reuse across deployments, and easing maintenance;
- improve the designer's confidence about the correctness, reliability, and performance of the deployed WSN application;
- are not developed in the abstract, rather take into account the resource limitations of the target devices and the concrete requirements of real-world applications, e.g., in terms of lifetime and robustness.

How do we achieve these goals? An obvious consideration is that they are germane to the techniques and expertise matured by software engineering (SE). Section 2 focuses on some of the core problems in the field, by concisely describing work already ongoing and opportunities for a contribution by the SE community. Nevertheless, the latter has been largely missing thus far: Section 3 provides some evidence for this statement and analyzes the current situation. Finally, Section 4 ends the paper by peering at what can be the future of a synergy between WSN and SE.

2. CORE OPEN ISSUES

As shown in Figure 1, WSN research today sorely misses support for the "low phases" of the development process. Indeed, the high phases (e.g., requirements and architecture specification) are much less biased by the peculiarities of WSNs: current SE techniques are by and large enough to carry out the job if and when needed. On the contrary, to allow WSN to make an impact in the real world, techniques and tools directly supporting developers are needed. The remainder of this section elaborates on two of the main open issues in the field: programming abstractions and methods for verification & validation of WSNs.

2.1 Programming Abstractions for WSNs: A Solved Problem?

The need for appropriate programming abstractions has been long-recognized by the WSN community, and several solutions are available. In a recent survey [7], we referenced 28 systems, which differ along many language and architecture dimensions, and therefore support a big fraction of the spectrum of application requirements. One could rightfully think that, given the number and variety of existing approaches, the issue of providing "good" programming abstractions for WSNs is a solved problem.

Unfortunately, a closer look shows that this is not the case. Of the 28 systems, only 13 have actually been implemented on WSN hardware: the others have been evaluated only through simulation. Moreover, to the best of our knowledge, only one—the TeenyLIME system [2]—has been used in a real-world deployment.

The (Concrete) Power of Abstraction. Is abstraction really worth in WSNs, where code size is small and resources scarce? This is a key question. In the following, we provide an answer biased by our own experience as part of the research team that developed TeenyLIME and built on top of it an entire long-term, real-world deployment for structural health monitoring of a medieval tower [1].

TeenyLIME [2] is based on the tuple space abstraction, a shared memory space where different processes read/write data in the form of tuples. Tuples are shared among nodes within radio range. In addition to common operations to insert, read, and withdraw tuples, reactions allow for asynchronous notifications when data of interest appears in the shared tuple space. In addition, several WSN-specific features are provided, e.g., to keep track of system-level information about neighbors. TeenyLime provides constructs useful to develop stand-alone applications as well as system level mechanisms, e.g., routing protocols, as demonstrated by our real-world deployment [1]. TeenyLIME is currently built atop TinyOS, targeting the TMote Sky platform.

The simple abstractions TeenyLIME provides are meant to replace the 1-hop message-based communication provided by nesC with 1-hop data sharing. The effects of this change in communication paradigm are many. From a qualitative point of view, the sharp decoupling provided by data sharing boosts reuse within and across applications, and leads to code that is inherently less complex. From a quantitative point of view, the code written by the developer is *significantly* more concise. The original TeenyLIME paper [2] reports that the KLOC size of a simplified sensor-actuator HVAC (Humidity, Ventilation, Air Control) application is 25-70% smaller with TeenyLIME than with nesC. These initial findings were confirmed by our real-world deployment [1] where we observed code reductions between 50% and 80% compared to alternatives in the TinyOS distribution.

A reduction in the size of the source code does not imply only a (beneficial) reduction of the programmer's effort in writing the code itself. It also yields smaller binary code, and therefore allows one to pack more functionality in the memory-tight WSN nodes. For instance, in our case we verified that we could not have fit the (simpler) alternatives provided by TinyOS for data collection, data dissemination, time synchronization in the 48 KB of code memory provided by the TMote Sky hardware—this despite the almost 10 KB consumed by TeenyLIME alone.

Therefore, in our experience, WSN programming abstractions are not a luxury: they are a *need*, in that they not only simplify development, but also enable an efficient use of program memory.

Programming Abstractions *for whom*? When designing a programming abstraction, a key requirement is *who* is the intended

user—an issue that did not receive enough attention thus far. We argue that two such users exist:

- *The "end user":* is generally a scientist, an engineer, or in any case the domain expert. Her main concern is to have "good data", e.g., without noise and accurately timestamped. For this user, the WSN is a macro-component delivering a useful service: she does not have good visibility of the WSN innards. This does not necessarily mean that the end user is uncomfortable with information technology: on the contrary, she is often quite skilled with basic IT tools (e.g., databases, spreadsheets, programming languages).
- *The "WSN geek":* is the person that, today, builds the software for, and often deploys, the WSN. The main concern of this user is to have "good yield", i.e., to funnel as much data as possible to the collection point, and to do so in the most efficient manner. This user has complete visibility of the WSN, as her objective is often to optimize the behavior of the individual nodes. Finally, this user is skilled at embedded systems programming and protocol design, and can deal with complex languages and systems, and their interaction.

From these short, informal definitions it appears that the two users have very different concerns and skills, which must be supported by different, dedicated abstractions. End users require high-level abstractions that simplify the configuration of the WSN at large, possibly allowing one to define its software architecture based on pre-canned components. WSN geeks need abstractions that, albeit lower-level, simplify the development of intra- and inter-node communication and computation. The answer to the question whether WSNs are going to become commonplace will be determined by the ability to empower the right user with the right programming abstraction, as many have already noted [3, 8].

2.2 Is My WSN Working?

Programming abstractions are important, as they simplify and rationalize the implementation of WSN software. However, as with every software artifact, development does not end at implementation: verification & validation must occur to increase confidence in the application correctness, and debugging may be necessary to fix the problems discovered. Contrary to programming abstractions, these are research areas where few contributions exist and, in contrast, where SE techniques have been used for a long time.

Debugging. WSNs complicate debugging significantly. The user interface of WSN nodes is limited to a few user buttons and LEDs. These, along with `printf` statements, are the building-blocks of state-of-the-art WSN debugging. The problem is further complicated by the fact that *i)* the resources on WSN nodes are scarce, therefore instrumentation is generally not viable; *ii)* the WSN is inherently distributed, which complicates the problem of observing and analyzing traces. A few works tackled the problem with different techniques, e.g., providing a `gdb`-like system [12] or relying on an auxiliary "sniffer" network used to check assertions on the WSN behavior [10]. However, at the time of writing, these works appear to have been used only for lab development: none has been reported as used in a real-world deployment.

Support for in-field debugging would be a very valuable asset for WSN developers, and is possibly the defining challenge for WSN debugging. Indeed, WSNs are often deployed in high numbers and in inaccessible places. These need not be military scenarios. As an example, in one of our projects [11] we are deploying about 90 WSN nodes in a road tunnel, to support energy-efficient lighting. Although this is a civilian application, access to the tunnel entails blocking partially or completely the traffic to ensure the safety of operators, and therefore must be minimized. This also means that it is often impractical to bring the nodes back to the lab, let apart the fact that the problem may be created by the real-world environment and not by the artificial conditions in the lab. Therefore, appropriate support for in-field inspecting the program state and possibly injecting changes is currently much needed.

Testing. A large SE literature on testing exists, along with a number of widely-used tools. However, very little of this knowledge has been transferred to WSNs. For instance, TUnit [6] provides a unit testing framework that focuses on improving the TinyOS distribution. Although in principle it could be used by any developer, its use in deployments has not yet been reported.

Moreover, WSNs pose specific challenges to testing. The peculiarities of WSN languages (e.g., the event-based, split-phase operation of nesC [4]) require adaptation of testing techniques, e.g., to assess test coverage and adequacy, or to analyze the code and automatically suggest tests. Moreover, WSNs are massively distributed systems, where each node must cooperate with the others. Tools like TUnit provide a simple answer for a single component on a single node, but say nothing about the distributed, coordinated behavior of the entire network. The latter, in turn, is strongly influenced by the environment where it is deployed. How to take realistically into account models or traces of the environment in terms of sensor readings, connectivity changes, or failures? The situation is paradoxical, in that this requires knowing environmental data, which is often the very purpose why a WSN is deployed. An incremental development approach, possibly integrated with simulation environments, may be a viable solution.

Formal Methods. Finally, as WSN applications become commonplace and encompass actuators for closing the control loop in-network, their provably-correct behavior becomes of paramount importance. SE techniques focusing on automatic verification (e.g., model checking) can play a fundamental role in increasing the confidence in the application correctness.

Similar problems have been already addressed in the context of embedded systems. However, *networked embedded systems*, as WSNs are often referred to, pose peculiar requirements in terms of scale, distribution, dynamicity and resource consumption. At the same time, however, they bring also opportunities. For instance, the code on WSN nodes is relatively small if compared to the one usually targeted by formal verification techniques. Therefore, the impact of state explosion—the enemy of model checking—is somewhat intrinsically reduced. Moreover, as we mentioned previously, languages and abstractions for programming WSNs are yet to be consolidated. Therefore, there is still an opportunity to design these abstractions by targeting not only programmer productivity, but also ease of automatic verification.

3. A HAPPY MARRIAGE OR A CONSENSUAL DIVORCE?

The challenges and opportunities listed so far clearly point in the direction of a fruitful synergy between the SE and WSN research fields—a happy marriage of different knowledge towards shared goals. SE is all about being systematic, disciplined, quantifiable, which is exactly what is needed in the WSN field today. By the same token, WSN is an eminently system-oriented, engineering-driven research field, which matches well the "engineering" angle of SE. SE can provide the WSN field with well-established tools, techniques and methodologies. In turn, their application in the novel domain of WSNs will enable SE researchers to find new research challenges and new solutions for old problems.

Unfortunately, the reality is more accurately represented through the metaphor of a consensual divorce, where the two fields acknowledge their mutual existence but in practice ignore each other. A symptom of this *status quo* can be obtained through a quick search on flagship SE scientific venues, such as ICSE, FSE, IEEE Trans. on Software Engineering (TSE) and ACM Trans. on Software Engineering and Methodology (TOSEM). In the last 4 years, these venues *together* published only one WSN-related paper, about testing a very specific aspect of nesC programs [5].

Another symptom we experienced directly. Along with a colleague, we submitted a tutorial proposal to ICSE'08 based on our aforementioned survey of WSN programming approaches [7]. The tutorial was accepted, only to be cancelled right before the conference because only 1 out of the nearly 900 attendees signed up for it. We had given a similar tutorial at other venues before and after that occasion, with a very different outcome. Aside from WSN conferences such as EWSN'09 and IPSN'10, we also held the tutorial at Middleware'08—a conference whose research community has significant overlapping with SE. In that occasion, our tutorial had over 30 attendees, out of the nearly 200 drawn by the conference. Although based on few observations, these considerations highlight a remarkable lack of interest[2] for WSNs in the SE community.

It is hard to tell why this is the case. The lack of interest may come from the fact that the SE community perceives WSNs as too "low-level". On one hand, this may lead researchers to think that the problems related to the development of software for tiny devices do not belong to mainstream SE. Or, on other hand, the fact that WSN software spans the entire network stack and reaches into the physical layer may constitute a steep learning curve for SE researchers. Another, less technical, reason may be the fact that there are not enough "bellwethers", i.e., senior, well-known researchers paving the road towards a synergy of the two fields, and somewhat highlighting its importance to the rest of the community. It should also be noted, however, that this is in line with the more general disaffection of the SE community for what we can call "modern distributed computing", i.e., pervasive, mobile, peer-to-peer computing. Although these topics are receiving a lot of attention by the research community at large, and contain elements that resonate with SE research, the SE community barely notices their existence. Very few papers on these topics are published, if at all, in SE venues.

Interestingly, the situation is quite the opposite when one considers the presence of SE topics in WSN venues. Debugging papers appeared at Sensys, IPSN, DCOSS. A couple of model checking papers published at IPSN'10 witness that there is a growing awareness in the WSN community that the field is mature enough to justify looking at principled ways of building WSN software. However, these SE papers are almost never authored by the people that dwell at flagship SE conferences. Therefore, although the need for SE techniques in WSN is evident, it is as if the two research communities are impermeable to each other—at least as of today.

4. DOES IT MATTER?

So, where does this leaves us?

We believe that the contribution of SE concepts to the WSN field is simply *inevitable*. Indeed, WSNs are a key element of the grand vision of a physical world augmented by a myriad of computing devices—known as Internet of Things, Cyber-Physical Systems, pervasive/ubiquitous/autonomic computing, ambient intelligence, etc. That this vision will be eventually realized is becoming more and more evident: the availability of increasingly powerful and smaller personal devices, along with the pervasiveness of sensing and computing devices in everyday objects (e.g., cars, washing machines) around us are clear signs of this transformation taking place. As a consequence, sooner or later, WSNs—as one of the enablers of this vision—are going to become mainstream. By then, they may be quite different from the WSNs we know today, e.g., with even smaller footprint and more powerful computing, different protocols, and so on. However, the fundamental challenges they pose, namely, enabling reliable and flexible coordination of large numbers of tiny devices, along with their in-field debugging and reconfiguration, will simply not go away. Becoming mainstream will mean essentially that enough of the tough "SE problems" in WSNs are solved, to enable reliable, cost-effective, easily reproducible and manageable deployments.

Whether this result is achieved by SE people is largely immaterial in the long run. However, joining forces between the SE and WSN research communities today surely appears like a win-win opportunity that is too attractive to miss.

[2]One reviewer of our ICSE'08 tutorial commented *"There may be a slight risk that it is a year or two too early, but this may be worth taking"*. With hindsight, the reviewer may have been right.

5. REFERENCES

[1] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corrà, M. Pozzi, D. Zonta, and P. Zanon. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *Proc. of the 8^{th} Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2009.

[2] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco. Programming wireless sensor networks with the TeenyLIME middleware. In *Proc. of the 8^{th} Int. Middleware Conf.*, 2007.

[3] Embedded WiSeNts Project—Research Roadmap. `www.embedded-wisents.org/dissemination/roadmap.html`.

[4] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proc. of the Int. Conf. on Programming Language Design and Implementation*, 2003.

[5] Z. Lai, S. Cheung, and W. Chan. Inter-Context Control-Flow and Data-Flow Test Adequacy Criteria for nesC Applications. In *Proc. of the 16^{th} Symp. on Foundations of Software Engineering (FSE)*, 2008.

[6] Rincon Research Corp. TinyOS 2.0 Automated Unit Testing. `http://www.lavalampmotemasters.com`.

[7] L. Mottola and G. P. Picco. Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art. *ACM Computing Surveys*, 2010. To appear. `disi.unitn.it/~picco/papers/surveywsn.pdf`.

[8] OnWorld—Emerging Wireless Research. `www.onworld.com`.

[9] G. P. Picco. Software engineering and wireless sensor networks: Happy marriage or consensual divorce? In *Proc. of the 1^{st} Int. Wkshp. on Software Eng. for Sensor Network Applications (SESENA'10), co-located with ICSE'10*.

[10] K. Römer. PDA: Passive distributed assertions for sensor networks. In *Proc. of the 8^{th} Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2009.

[11] TRITon. Trentino Research & Innovation for Tunnel monitoring. `triton.disi.unitn.it`.

[12] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: A comprehensive source-level debugger for wireless sensor networks. In *Proc. of the 5^{th} Int. Conf. on Embedded Networked Sensor Systems (SENSYS)*, 2007.