Swift Fox whitepaper

Marcin Szczodrak [*], Vasileios P. Kemerlis [†],

Xuan Linh Vu [‡] and Yiwei Gu [§]

Computer Science Department
Columbia University
New York, NY

Manuscript received at February 24, 2010; revised September 30, 2011

1 Introduction

For over a decade, *wireless sensor networks* (WSNs) fostered the development of new applications and trends in the broad areas of mobile and opportunistic computing. Such networks are composed of multiple small, low-cost and low-power, multipurpose sensor nodes that communicate with each other over small distances using the wireless medium [5]. Their *self-organizing* capabilities, due to their ad hoc communication paradigm, make them suitable for deployment on demand. This saves engineering time, reduces the installation costs, and accommodates the ad hoc deployment of a wireless network during disaster emergencies or military operations. Hence, applications of WSNs are pervasive in military (*e.g.,* ammunition monitoring, battlefield surveillance, sniper localization [27], reconnaissance, damage assessment), environment (*e.g.,* animal tracking [29], chemical detection [2], pollution monitoring [15]), health (*e.g.,* patient monitoring and drug surveillance in hospitals) [25], home automation (*e.g.,* "smart home") [21], as well as in the commercial domain (*e.g.,* monitoring construction material, inventory management, building automation, vehicle tracking).

[*] msz@cs.columbia.edu
[†] vpk@cs.columbia.edu
[‡] xv2103@columbia.edu
[§] yg2181@columbia.edu

Figure 1 illustrates the architectural components of a typical sensor node (also known as "mote"). Each mote consists of a *micro-controller*, an outward *power source*, a *transceiver*, some amount of *external memory* (*i.e.,* external in terms of the micro-controller), and a set of *sensor devices* all connected with the micro-controller through an *analog-to-digital converter* (ADC). Each node senses its external environment and conveys the perceived data to nearby nodes using the IEEE 802.15.4 wireless communication standard. Moreover, special purpose nodes act as *base stations* and collect all sensor data in order to further deliver them to aggregation points. That is, they act as "bridges" between network boundaries, or help in establishing a multi-tier network topology [12].

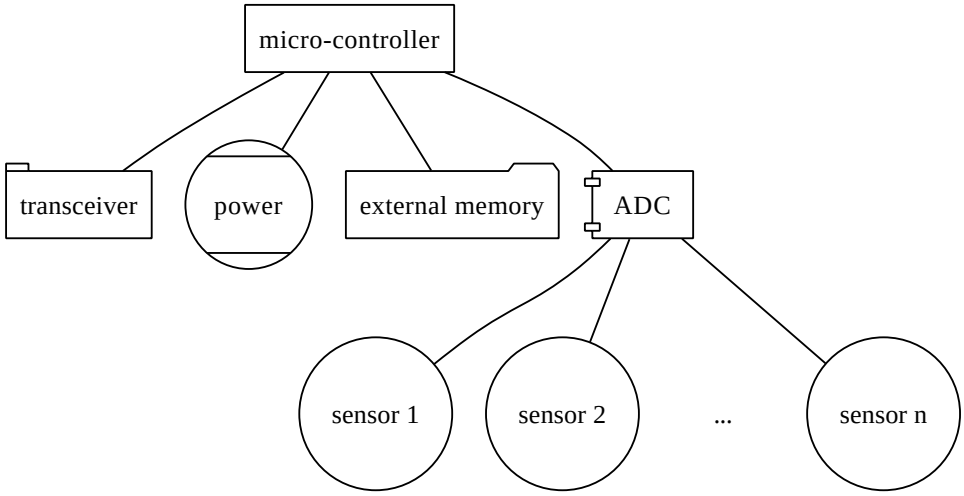

Figure 1: Sensor node (*mote*) architecture.

In the early 2000's, WSNs were envisioned as the materialization of the "Smart Dust" concept (*i.e.,* millimeter scale sensing and communication platforms) [30]. However, due to power consumption issues we have yet to witnesses nodes of such scale; instead of "dust" sensors, we had only sensors at the size of a small PDA. Having said that, recent advances on the sensor chips exploit *ambient* power (*e.g.,* radio power from the TV and FM radio) in order to further miniaturize their size by avoiding batteries. Moreover, they offer accurate estimates of the processing that can be performed with certain amount of consumable energy, fueling even more the development of new services of *ubiquitous* manner. As a characteristic example consider the Central Nervous System for the Earth (CeNSE) project, which has been

announced by HP and aims at the development of trillion nano-scale sensors and actuators that will all be interconnected with computing systems and provide new services [19].

2 Problem statement

Akyildiz *et al.,* [1] surveyed a large number of different WSN applications and identified a set of factors that affect the design of a sensor network. Such factors include fault tolerance, scalability, operating environment, network topology, hardware constraints, transmission media, and various others. This variety of aspects that deeply affect the design, and therefore the implementation and operation, of a WSN, gave rise to an enormous number of WSN applications that heavily re-implement basic system facilities (*e.g.,* network protocols, routing, data dissemination and collection) in order to provide optimal results. This style of development although it offers many degrees of freedom for the design and development of new and elaborate facilities, it has also serious drawbacks; it delays the adoption of the WSN technology by non-experts (*i.e.,* a typical vendor that provides WSN software and services should have deep understanding and expertise in various fields, ranging from OS development to routing protocol design and implementation).

Fennec Fox is a platform that facilitates policy-based, self-controlling, dynamic network *reconfiguration* of low-power and lossy networks (*e.g.,* WSNs). It aims at maximizing the efficiency of the system by managing the provided resources, appropriately, so as to meed application requirements at the lowest possible cost. The WSN operation is constrained by power, memory, and computational resources. Sensor nodes are typically battery powered, with few kB to few MB of memory, and have limited processing power (*e.g.,* their clock speed ranges from kHz to few MHz). Using these limited resources, WSNs are expected to support various applications. The Fennec Fox platform is the apparatus of a philosophy, which assumes that WSN applications consist of a *chain of network tasks.*

A task is defined as a simple computational operation, optimized to perform some functionality given certain constraints (*e.g.,* power consumption, delay, security). During their life-cycle, Fennec Fox-based sensor nodes *transition* between various states, where each state performs a simple task. In this context, applications running on Fennec Fox identify conditions for transitioning between various tasks, and controlling the behavior of the tasks, by specifying task parameters. However, to perform their job, tasks require

support from other parts of the WSN that provide various services, such as routing or sensing. Therefore, sensor states are not only represented by the task which they perform, but also by a set of other WSN systems that help this task to achieve its expected results.

Fennec Fox was motivated by the need to *auto-configure*, *reprogram*, and *manage* wireless sensor networks. Since such tasks are already complicated by themselves, Fennec Fox attempts also to simplify the modeling and programming of sensornets. That is, sensornets are required to auto-configure themselves during their lifetime. [1]

Fennec Fox is built out of three logical components: a *dynamic network stack*, a *policy control unit*, and a *user programmable application layer*. The dynamic network stack is a library, consisting of protocols and utilities for providing a high level abstraction to the TinyOS [16] system, which supports adaptive applications. Policy control unit is the component responsible for guiding the transitioning between different stack configurations. The transitions are expressed in form of policies, which in response to an event, reconfigure the stack. Finally, the application layer provides a simplified API; it allows the programmer to focus on the application logic, instead of reprogramming communication protocols or mathematical functions. Swift Fox language is a high-level "handle" for programming Fennec Fox. The following example demonstrates the usage of Fennec Fox and Swift Fox and defines their relationship. Suppose that we have a WSN for home monitoring that is also capable of handling emergency situations. In this setup, our network application supports two scenarios; the first one is focusing on monitoring the home environment by collecting sample data from all sensor nodes, and the second one is focusing on monitoring areas with the potential of fire. For the needs of the second scenario, the WSN is capable of taking and streaming pictures. Yet, collecting sensor data, such as light intensity, humidity, and temperature, requires network protocols that can support the type of network traffic generated by sensor nodes, as well as a network topology that can efficiently disseminate the collected data. A simple representation of data collection scenario is illustrated on Figure 2. In the second scenario, the sensors detect a fire, based on multi-modal correlation of sensor readings that represent increased probability of fire occurrence. In that case, when fire is detected, we want the system to capture pictures of the

[1] Ali El Kateeb has stated that "because wireless sensor network applications are new and might require abilities not fully anticipated during design and development, supporting sensor networks with design-modification capabilities is vital to making sensor-network platforms capable of adjusting to their surrounding environment after initial network deployments." [20]

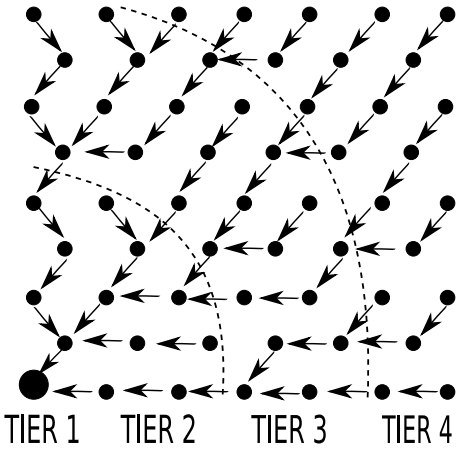

Figure 2: Sensor network that establishes a multi-tiered tree topology, where data are propagated toward higher tiers (lower in id), and eventually reach a sink node. To optimize network performance, for example to lower power consumption, data may be aggregated or compressed to decrease the size of the transmitted messages.

fire area to verify the correctness of the fire alert and also as estimate if there are still people around the fire area who can be in danger.

From the network communication perspective, this scenario requires establishing *point-to-point* communication between a sensor node, which can capture a picture of the area in fire, and the sink node, for providing direct high bandwidth communication for data transmission in an emergency situation such as this one. A representation of network configuration supporting this scenario is shown in Figure 3. Both figures show scenarios with different data flow objectives: *network energy conservation* versus *low-delay direct data transmission.*

Fennec Fox is a platform that provides the ability to have a reconfigurable WSN that adapts to different application requirements. Through Swift Fox we want to be able to program the Fennec Fox platform, seamlessly, in a high-level and abstract fashion. Hence, Swift Fox is a simple policy-based programming language that specifies conditions that have to be meet for a network to be reconfigured. Using the fire emergency application, we want Swift Fox to reconfigure the network from the monitoring scenario to the emergency scenario whenever light sensors show increased light intensity, humidity sensors show decreased humidity, and temperature sensors show

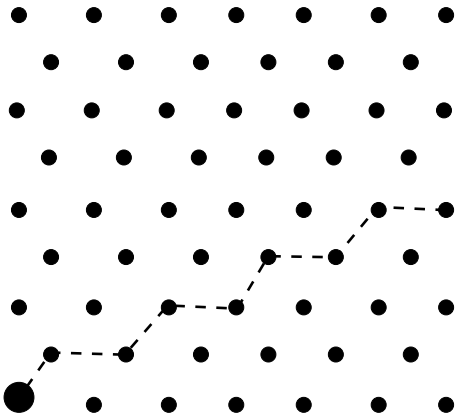

Figure 3: In a result of some event, a point-to-point communication between a node and the sink node is established. Network topology is flattened, no data aggregation is performed, and low delays are imported.

an increase in temperature. Let `monitor` and `fire` represent network state configurations that correspond to the two aforementioned scenarios (*i.e.,* the home monitoring environment and the fire emergency response). Then, using Swift Fox language we would like to be able to express a condition, represented by an event e captured by the sensors, which would trigger a transition on the network configuration from `monitor` state to `fire` state.

We define eleven objectives that Swift Fox should provide in order to become a successful programming language and we specify them in a form of "buzzwords" as follows:

1. **Policy oriented**
   Policies describe precisely the behavior of entities that reside within the bounds of a *policy domain*. Swift Fox allows, implicitly, the definition of a policy domain by giving the ability to describe, seamlessly and accurately, reconfiguration scenarios in an intuitive way. In general, each entity (*i.e.,* a sensor node) is capable of responding differently to certain stimuli, based on the actual facilities provided by the underlying reconfiguration platform (*i.e.,* Fennec Fox). For example, given the sensation of a certain event that indicates a critical incident, a node can cause the reconfiguration of the whole network in order to best disseminate critical data (*e.g.,* proof of destruction or imminent hazard). Evidently, not all possible responses are permissible, either for security reasons (*e.g.,* a node that does not have a vibration sen-

sor should not be able to reconfigure the network for an earthquake response) or for application-specific purposes (*e.g.,* there might be a strict priority among different applications based on their resource consumption). Through Swift Fox the task of defining a *deterministic* and *authoritative* behavior for the WSN, as a whole, is largely simplified and becomes less error-prone. This is accomplished by adopting a policy-based notation that captures the essence of the *environment semantics* as well as the entities' *rights*, *prohibitions* and *obligations*.

2. **Secure**
   WSNs communicate and provide their services over the wireless medium. The open nature and the omnipresence of wireless signals, along with the lack of physical barriers in most WSN installations, leave a window of opportunity open for the proliferation of malicious behavior (*i.e.,* either because WSNs provide an infiltration channel to a certain domain or because the WSN is the target itself). Therefore, protection against malevolent adversaries will most likely be mandatory in the near future. Authentication protocols, access control mechanisms, and encryption schemes have been studied in the past (in the WSN context) and have already lead to established techniques that offer specific levels of *assurance*. However, most of the available solutions are highly optimized for certain scenarios. Hence, it is of paramount importance to allow dynamic configurations that enable the reuse of *trusted*, mature, and provable-working security components. Swift Fox offers the potential for expressing, naturally, specific security guarantees as well as behavioral requirements for the WSN nodes. Moreover, it allows the combination of different security primitives (*i.e.,* distributed authentication, key exchange, encryption) in a composable fashion that meet the needs of the environment and serve the applications to the best extent possible.

3. **Robust**
   In Section 1, we outlined some of the possible WSN usages and application scenarios. However, many of these tasks (*e.g.,* battlefield surveillance, patient monitoring, inventory management) are critical because they might either affect human lives or hinder assets that are essential for the efficient operation of other systems. In particular, for such tasks the WSN is not allowed to exhibit *best-effort* behavior and it is considered a *critical infrastructure*. Swift Fox provides *error-resistant*, *reliable* WSN configurations by allowing the deterministic

expression of WSN states in a high-level fashion.

4. **Simple**
   Swift Fox is designed to fuel the development of commercial WSN applications. Thus, we assume that the audience of the language, people that deploy WSNs or simple contractors, may not be aware of every underlying aspect of a WSN. Yet, Swift Fox should be able to provide a simple way of programming a WSN by defining reconfiguration policies.

5. **Service Oriented**
   Swift Fox provides a way to express stages in the lifetime of a WSN. Its service oriented architecture (SOA) is inherited from Fennec Fox platform, which provides network services for multiple applications. Swift Fox loosely integrates Fennec Fox services with their applications into a network program. This network program is a chain of network states, during which one application with its supporting network protocols is active, and transitions between network states occur in response to events sensed by the network.

6. **Declarative**
   The management of WSNs requires well-defined declarative languages [18]. The programmer should be able to describe *what* should the network configuration be in every case, via event handling actions. In contrast to a procedural language (like C), where the program is a set of step-by-step instructions for execution, a declarative language aims at providing assertions on beginning conditions and destination targets [17]. Nevertheless, like any programming language, it must be *expressive* enough so both the programmer and the compiler to be able to grasp how the program performs over time.

7. **Domain specific**
   Swift Fox is designed specifically for programming and managing WSNs. Its usage scope is well defined for the middle layer of network management. Our language belongs to the same range of domain specific languages (DSLs), in the WSN context, like nesC and Verilog. In fact, at the moment of writing, nesC is the underlying language that Swift Fox compiles into. As a DSL, Swift Fox is developed to address the needs of a given domain and therefore it can only cater to a limited number of concepts. This leads to a higher level language that improves the developers' *productivity* and *communication* with domain

experts [6]. The typical characteristics of a domain specific language are the following: it is designed for a particular problem domain (in our case WSNs), it is intended to glue together with other languages to complete a system (*e.g.,* nesC), and finally it resembles configuration code (*i.e.,* policy specifications) [10]. Moreover, keeping the language confined allows for use of libraries, which are pre-compiled in whatever languages they need, in order to complete the desired underlying tasks. The configuration part of the language can be *expressive* and easily *readable*.

8. **Lightweight**
   Software subsystems can often be designed and implemented in a *clear*, *succinct*, and *aesthetically pleasing* way, by using specialized linguistic formalisms. In cases where such formalisms are not compatible with the principal implementation language (*e.g.,* nesC), lightweight languages [28] are necessary so as to provide the required primitives as meta-constructs. The minimalistic syntax of Swift Fox not only allows memory-efficient objective code, but also provides the ability to exploit the primitives of the underlying platform in an clear, straightforward, and coherent manner.

9. **Network-savvy**
   Swift Fox provides a computing abstraction for the TinyOS system and Fennec Fox platform, which contain libraries of various protocols, such as medium access protocols (MAC), network protocols (*e.g.,* TCP/IP [7] and IPv6 [9]), data dissemination protocols (*e.g.,* Trickle[23]), quality of information (QoI) protocols, reconfiguration schemes (*e.g.,* TENET [12] and Mate [22]), and on demand sensor addressing [26]. Swift Fox allows the programmer to adequately match the services provided by various protocols to quality of service (QoS) expectations imposed by applications.

10. **Platform independent**
    Swift Fox compiles into nesC [11] that is suported by the Fennec Fox platform running on top of TinyOS. In some way, this approach resembles Java; Java code is compiled into byte code for Java Virtual Machine (JVM). The same way as Java becomes platform independent through JVM, Swift Fox becomes independent through Fennec Fox. Currently, Fennec Fox supports only TinyOS; however, in future other popular embedded systems are expected to be supported, particularly Linux, and Contiki [8], which have strong commercial support.

Other operating systems, such as MANTIS [3], LiteOS [4], SOS [14], and Pixie OS [25] will be supported as time will allow. Today, wireless sensor nodes running all these operating systems create separate networks, with different programming environments, and with various programming interfaces. Swift Fox establishes a single application programming interface and unifies various, currently, disjointed network environments. Moreover, it greatly simplifies WSN deployment and management, and encourages WSN production and deployment from various vendors.

11. **Event-driven**
    Swift Fox belongs to a family of languages that express system policies, and compiles into the nesC language [11], a component-based event-driven programming language. The event-driven nature of sensors becomes more evident as hardware reconfiguration mechanisms get to be incorporated in the sensor nodes [20]. The event-driven nature of Swift Fox stems from the *event-condition-action* (ECA) information model of polices, which for the first time proposed and implemented in Bell Labs in 1999, and called Policy Description Language (PLD) [24]. As shown in PLD, this information model of policies allows the creation of a programing language that does not have to be based on any graphical representation (*e.g.,* in Unified Modeling Language (UML)). Finally, Swift-Fox establishes a relation between the event-driven specification of an embedded system and the event-driven policy that describes WSNs reconfiguration.

3 Language overview

Swift-Fox like many other languages, such as AWK, "was born from necessity to meet a need" [13]. Similarly to AWK, we are aiming for a simple language that we can easily use to express transitions between the states of a WSN platform. These transitions occur in response to events detected by the sensor nodes. Here we propose a simple *event-condition-action* language – Swift Fox – that we hope will meet our expectations. Because we believe that we are the first one to address the wireless sensor network reconfiguration problem by designing a new programming language, we hope this language will become successful outside the walls of the classroom. To make this a successful case, we defined eleven characteristics which we want our language to posses: policy oriented, secure, robust, simple, declarative,

service oriented, domain specific, light weight, network savvy, platform independent, and event driven.

References

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[2] Jude Allred, Ahmad Bilal Hasan, Saroch Panichsakul, William Pisano, Peter Gray, Jyh Huang, Richard Han, Dale Lawrence, and Kamran Mohseni. SensorFlock: An airbone wireless sensor network of micro-air vehicles. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 117–129, Sydney, Australia, November 2007.

[3] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *ACM/Kluwer Mobile Networks & Applications (MONET)*, 10:563–579, 2005.

[4] Qing Cao, Tarek Abdelzaher, John Stankovic, and Tian He. The LiteOS operating system: Towards unix-like abstractions for wireless sensor networks. In *International Conference on Information Processing in Sensor Networks (IPSN)*, pages 233–244, St. Louis, MO, USA, April 2008.

[5] David Culler, Deborah Estrin, and Mani Srivastava. Guest editors' introduction: Overview of sensor networks. *Computer*, 37:41–49, 2004.

[6] Johan den Haan. 7 recommendations for domain specific language design based on domain-driven design. Online, May 2009.

[7] Adam Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of the 1st ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 85–98, San Francisco, CA, USA, May 2003.

[8] Adam Dunkels, Bjrn Grnvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings*

of the 1st IEEE Workshop on Embedded Networked Sensors (Emnets-I), pages 455–462, Tampa, FL, USA, November 2004.

[9] Mathilde Durvy, Julien Abeillé, Patrick Wetterwald, Colin O'Flynn, Blake Leverett, Eric Gnoske, Michael Vidales, Geoff Mulligan, Nicolas Tsiftes, Niclas Finne, and Adam Dunkels. Making sensor networks IPv6 ready. In *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, 2008. poster session.

[10] Martin Fowler. Introduction to domain specific languages. Online, October 2006.

[11] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to network embedded systems. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 1–11, San Diego, CA, USA, June 2003.

[12] Omprakash Gnawali, Ben Greenstein, Ki-Young Jang, August Joki, Jeongyeup Paek, Marcos Vieira, Deborah Estrin, Ramesh Govindan, and Eddie Kohler. The TENET architecture for tiered sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 153–156, Boulder, Colorado, USA, October 2006.

[13] Naomi Hamilton. The A-Z of programming languages: AWK. Online, May 2008.

[14] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the 3rd ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 163–176, Seattle, WA, USA, June 2005.

[15] Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. FireWxNet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 28–41, Uppsala, Sweden, June 2006.

[16] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35:93–104, 2000.

[17] Jing Huang. Introduction to semantics of programming languages. Online, September 1994.

[18] John Hughes. Why functional programming matters. *The Computer Journal*, 32:98–107, 1989.

[19] Information and Quantum Systems Labs – HP. Central nervous system for the earth (CeNSE). Online, February 2010.

[20] Ali El Kateeb. Hardware reconfiguration capabilities for third-generation sensor nodes. *Computer*, 42:95–97, 2009.

[21] Younghun Kim, Thomas Schmid, Zainul M Charbiwala, Jonathan Friedman, and Mani B Srivastava. NAWMS: Nonintrusive autonomous water monitoring system. In *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, pages 309–322, Raleigh, NC, USA, November 2008.

[22] Philip Levis and David Culler. Mate: A tiny virtual machine for sensor networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, USA, October 2002.

[23] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, San Francisco, CA, USA, March 2004.

[24] Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. A policy description language. In *Proceedings 16th National Conference on Artificial Intelligence (AAAI) and 11th Conference on Innovative Applications of Artificial Intelligence (IAAI)*, pages 291–298, Orlando, FL, USA, July 1999.

[25] Konrad Lorincz, Bor rong Chen, Jason Waterman, Geoff Werner-Allen, and Matt Welsh. Resource aware programming in the Pixie OS. In *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys)*, pages 211–224, Raleigh, NC, USA, November 2008.

[26] Curt Schurgers, Gautam Kulkarni, and Mani B. Srivastava. Distributed on-demand address assignment in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 13:1056–1065, 2002.

[27] Gyula Simon, Miklos Maroti, Akos Ledeczi, Gyorgy Balogh, Bronislav Kusy, Andras Nadas, Gabor Pap, Janos Sallai, and Ken Frampton. Sensor network-based countersniper system. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–12, Baltimore, MD, USA, November 2004.

[28] Diomidis Spinellis and V. Guruprasad. Lightweight languages as software engineering tools. In *Proceedings of the Conference on Domain-Specific Languages (DSL)*, pages 67–76, Santa Barbara, CA, USA, January 1997.

[29] Robert Szewczyk, Eric Osterweil, Joseph Polastre, Michael Hamilton, Alan Mainwaring, and Deborah Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47:34–40, 2004.

[30] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S.J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *IEEE Computer*, 34:44–51, 2001.