Archetype-Based Design: Sensor Network Programming for Application Experts, Not Just Programming Experts

Lan S. Bai† Robert P. Dick† Peter A. Dinda‡
{lanbai, dickrp}@umich.edu pdinda@northwestern.edu
University of Michigan† Northwestern University‡

ABSTRACT

Sensor network application experts such as biologists, geologists, and environmental engineers generally have little experience with, and little patience for, general-purpose and often low-level sensor network programming languages. We believe sensor network languages should be designed for application experts, who may not be expert programmers. To further that goal, we propose the concepts of sensor network application archetypes, archetype-specific languages, and archetype templates. Our work makes the following contributions. (1) We have examined a wide range of wireless sensor networks to develop a taxonomy of seven archetypes. This taxonomy permits the design of compact languages that are appropriate for novice programmers. (2) We developed a language (named WASP) and its associated compiler for a commonly encountered archetype. (3) We conducted user studies to evaluate the suitability of WASP and several alternatives for novice programmers. To the best of our knowledge, this 56-hour 28-user study is the first to evaluate a broad range of sensor network languages (TinyScript, TinySQL, SwissQM, and TinyTemplate). On average, users of other languages successfully implemented their assigned applications 30.6% of the time. Among the successful completions, the average development time was 21.7 minutes. Users of WASP had an average success rate of 80.6%, and an average development time of 12.1 minutes (an improvement of 44.4%).

1. INTRODUCTION

Wireless sensor networks have the potential to allow inexpensive, real-time, and broadly-distributed data collection and analysis for the first time in history. Application experts, such as geologists, entrepreneurs, civil engineers, and biologists have the most to gain. However, these application experts generally have little experience with, and little patience for, the general-purpose, node-level languages available for sensor network programming. As a result, application experts are forced to rely on embedded system experts to implement their ideas. Almost all existing sensor network deployments are implemented by embedded system experts. This approach is costly. Separating design and implementation in this way can also lead to errors due to miscommunication between application experts and embedded system experts. Application experts generally have limited awareness of the constraints on sensor network capabilities imposed by hardware and software limitations. On the other hand, embedded system experts know little about the application requirements, which are tightly related to the measured objects and the working environments. In addition, since application experts' and embedded system experts' domain languages differ significantly, this can cause confusion and misunderstandings that lead to incorrect implementations. Consequently, a collaboration between application experts and embedded system experts requires a large amount of communication, negotiation, redesign, and reimplementation. Many potential users of wireless sensor networks consider them because they have the potential to save time and money. When these potential benefits are outweighed by substantial increases in implementation complexity compared to the bulky and expensive, but often easy-to-deploy, sensing solutions already in use, wireless sensor networks will remain unused.

We believe that appropriate high-level programming languages and compilers have the potential to make wireless sensor networks accessible to the application experts who have the most to benefit from their use. We propose designing sensor network languages with the novice programmer in mind, hence the following language features are desirable.

1. The languages should support specifying application-level requirements, not just node-level behavior.

2. The languages should not expose low-level implementation details, such as resource management, communication protocols, and optimizations, to users. Users

IPSN'09, April 13–16, 2009, San Francisco, California, USA.

should need only specify application requirements.

3. The languages should be compact and easy to use. People with limited or no programming experience should be able to almost immediately learn and use them to specify correct sensor network applications.

The first step to designing a programming language is to determine the scope of applications it will support. There are two extremes of the range of design philosophies a language designer might adopt: a language might be entirely general-purpose or entirely application-specific. General-purpose languages can be used to specify any application. However, all other things being equal, this flexibility is obtained at the cost of increased language complexity. General-purpose languages have advantages: once such a language is learned, one can write any application with it. However, a novice programmer may never be willing to expend the time to learn it. In contrast, application-specific languages are usually simple and compact, but support one type of application. This makes it more difficult for a novice programmer to select the appropriate language for an application, and requires the design of numerous languages – one for each type of application. Designers need to learn a new language with each new application. We believe the optimal design philosophy for sensor network programming languages is somewhere between these extremes: a moderate number of specialized languages that together cover most of the sensor network application domain. Ideally, each of these languages should be easy to learn and use for novice programmers.

The question remains, "what is the appropriate granularity of the application groups?", i.e., "how specialized should the languages be?" To find the best tradeoff between the complexity of selecting a language and the complexity of the languages, we propose the concept of *sensor network archetypes*. We have categorized sensor networking applications into archetypes based on functional properties that have large impacts on language design. We have examined a wide range of sensor network applications in order to develop a taxonomy of seven archetypes (see Section 3). The language tailored for an archetype is called an *archetype-specific language*.

Once an application's archetype is known, it is possible to provide a program template/example as a starting point. Our studies indicate that the availability of templates improves the success rate for novice programmers implementing sensor network applications from 0% to 8.3% for a node-level language. Knowledge of an archetype further reduces the burden on a novice programmer because only one archetype-specific language needs to be learned, and each such language is simpler than a general-purpose programming language. We have embodied these language design concepts in a language for a frequently encountered sensor network arche-type. In comparison with alternative sensor network programming languages such as TinyScript, TinyDB, and SwissQM, this language results in 1.6× average improvement in success rate and 44.4% average reduction in development time. We plan to extend these language design concepts to other archetypes in our future work.

2. RELATED WORK

A number of researchers have proposed new sensor network languages to improve design productivity. However, most of these languages have been designed with expert programmers in mind. Although they may improve the productivity of embedded system experts, they are unlikely to make the design and deployment of sensor networks accessible to application experts who are often novice programmers. A few languages are proposed for application experts. However, their use by novice programmers has not been experimentally evaluated, making it difficult to draw conclusions about their suitability. In this section, we review these languages and summarize the major differences and contributions of our work.

Node-level programming languages specify the behavior of each single sensor node. To access the state of other nodes, the program must explicity specify data transmission between nodes. NesC [11] and C are widely used node-level programming languages for sensor networks. These languages are too low-level for novice programmers. In addition, concepts such as events and threads are quite difficult for novice programmers to learn. Efforts [14, 24] have been made to raise the abstraction level of these languages.

Network-level programming languages, also called macro-programming languages, treat the whole network as a single machine. Lower-level details such as routing and communication are hidden from programmers. Pleiades [19] extends C to achieve a centralized perspective with access to all the nodes in the network via naming. Some researchers allow designers to treat the sensor network as a database and use query languages to extract data from the network [28, 33]. Regiment [34] lets programmers view the network as a set of distributed data streams. ATaG [4] is based on data-driven program flow and mixed imperative-declarative specification. It lets developers graphically declare the data flow and connectivity of virtual tasks and specify the functionality of tasks using common imperative language. RuleCaster [5] provides a macroprogramming abstraction with a state-based model and uses a high-level language akin to Prolog.

A few researchers have considered the accessibility of sensor network design to application experts. However, we are aware of only one other publication describing experiment evaluation of usability of a sensor network programming language. Some languages [12, 17] are inspired by commercial graphical programming tools such as LabView [21] and Excel [10]. Other researchers made the design of easy-to-use languages tractable by targeting a specific type of applica-

tions. NETSHM [7] is a sensor network software system for structural health monitoring applications. In contrast, we did a broad study of existing sensor network applications and proposed a method of identifying classes of applications for specialized language development. BASIC was proposed for use in sensor network programming [30]. The authors implemented BASIC for sensor networks and conducted a user study with novice programmers. Their user study is contemporaneous with ours. Their work targeted a different application domain than ours and focused on node-oriented programming.

More comprehensive reviews and comparisons of existing sensor network programming languages can be found in surveys [46, 31]. Sugihara and Gupta [46] compared the languages using three metrics: energy-efficiency, scalability, and failure-resilience. They acknowledged that ease of programming is a very important criteria but they believed "criteria of easiness is inherently subjective and the complexity of code largely depends on each application". In contrast, we believe that it is possible and important to evaluate the usability of sensor network languages and have designed and executed a rigorous user study to compare a number of languages. Mottola and Picco [31] introduced a taxonomy of wireless sensor network programming models. The taxonomy we introduce is for sensor network applications. Röemer and Mattern [38] have studied and characterized the design space of sensor networks to provide a clearer picture of sensor network applications and their requirements. We also classified sensor network applications but for a different purpose: archetype-based programming language design. We focus solely on dimensions that affect the complexity of specification language.

Our work shares some techniques and principles with that on human–computer interaction [13, 20] and the psychology of programming [49]. Involving users in the design process has been emphasized in the human–computer interface design.

3. TAXONOMY OF WIRELESS SENSOR NETWORK APPLICATIONS

Specialized, high-level specification languages have the potential to open sensor network design to application experts who are novice programmers. Finding the optimal partitioning of the sensor network application domain for the purpose of language design is challenging. This section describes our study of a wide range of sensor network applications in order to build a taxonomy of sensor network archetypes, and thus languages.

3.1 Wireless Sensor Network Applications

We studied 23 sensor network applications and summarized their application-level requirements and functionalities to extract 19 application properties. These applications, most of which have been deployed, span a wide range of domains: environmental monitoring [16, 47, 50, 44, 40], structural health monitoring [8, 6, 26, 45, 35, 18], habitat monitoring [37, 27], target detection and localization [15, 39, 2, 43], residential monitoring [51], active sensing [52], medical care [41], farm management [42, 48], and others [9]. Specifications should focus on the requirements of an application, and avoid implementation details to the greatest degree possible while still maintaining adequate performance. Based on this principle, we identified the following 19 application-level properties (refer to Section 3.2 for definitions): mobility, initiation of sampling process, initiation of data transmission, interactivity, data interpretation, data aggregation, actuation, homogeneity, topography, sampling mode, when sensor locations are known, synchronization, unattended lifetime, mean time to failure, maximum node weight, maximum node size, maximum node volume, maximum node mass, minimum covered area, and quality of service.

3.2 Categorization of Wireless Sensor Network Applications

Among the 19 application properties, only eight affect the complexity of the specification language. Other properties are constraint oriented and have little impact on the specification of sensor network functionality. For example, changing the required lifetime of the system from a month to a year will not change the functional specification, although the implementation may change. Specifying constraints can be uniform and straightforward across many application domains, unlike functional specifications. The syntax *constraint* = *value* is sufficient. Therefore, we ruled out these properties as criteria for placing applications. The following eight properties remain.

- **Mobility** indicates whether the sensor nodes are mobile. Mobile nodes may be wearable devices to monitor or track moving objects such as humans and animals [48, 42]. Sensor nodes might also adjust their positions. For applications with mobile sensor nodes, specifications of node localization and node movement control are usually desired. Therefore, mobile sensor network applications will require more complex specifications.

- **Initiation of sampling** indicates the condition that causes the nodes to start sampling. It can be periodic, event driven or a mix. Periodic sampling requires specification of the sampling period, while event-driven sampling requires the specification of events.

- **Initiation of data transmission** indicates the condition in which nodes send data through the network. It can be periodic, event driven, or both. Applications for