Event Detection In Wireless Sensor Networks

Krasimira Kapitanova, Sang H. Son
Department of Computer Science
University of Virginia
{krasi, son}@virginia.edu

Abstract

As the demand to explore the physical world increases and the sensing and activation tasks become more sophisticated, wireless sensor networks are expected to support more complicated applications. Regardless of the specific application running on the network, sensors should be able to individually or collaboratively provide diverse event services, i.e. report the occurrence of different events of interest. We present our current work on developing a formal event specification language, designed specifically for sensor networks, and a specification transformation approach that enables the translation of formal event specification into code.

1. Introduction

Event detection is one of the main components of many wireless sensor network (WSN) applications. Sensor networks are used in health monitoring applications to detect abnormal patient behavior, military surveillance WSNs are deployed to detect suspicious enemy activity, fire fighting sensor networks are deployed to set an alarm if a fire starts. Regardless of the specific application, the network should be able to detect if particular events of interest have occurred. However, effective approaches to specifying events remain a challenge.

The need for a description language that can incorporate the knowledge of sensing with event definitions is evident. A formalized description language can serve as an effective interface between people who register events, e.g. application semantics experts, and sensor network designers. Such a language could also help clarify the application requirements and remove the ambiguity often seen in application specifications. Here we present our on-going research on developing an event specification language for WSNs. The foundation of our work is MEDAL [4], an extended Petri net that can capture the structural, spatial, and temporal properties of a complex event detection system. MEDAL can also be used as an analysis tool to help system designers identify inconsistencies and problems in their applications before and after deployment of the sensor network.

Specifying the events the sensor network should recognize is an important component of event detection. However, once the events have been specified, the formal model should be translated into code that can be executed by the nodes. To address this, we have designed an architecture for a full-fledged event detection service that transforms the formal models into executables.

2. Event specification

Event specification for WSN applications is greatly influenced by the nature of sensor networks. WSN applications are distributed, concurrent, and rely on the use of numerous resource constrained sensor nodes. These nodes are generally considered unreliable, and their readings are often imprecise and noisy. Therefore, there is a lot of randomness and non-determinism associated with event detection in WSNs. In addition, events are a function of when and where they occur, and they often require collaborative decision making which is based on the spatial and temporal composition of the sensor readings. This places communication in the heart of event detection. Consequently, it also becomes an important part of event specification.

Most papers that use event definitions in sensor networks employ SQL or SQL-like semantics to describe the events [1, 2, 5, 6]. However, as pointed out in [7], SQL-like semantics are not always suitable for sensor networks because of the lack of collaborative decision making and other necessary features. The inadequacy of SQL in specifying WSN events comes from some of the essential characteristics of sensor network events: (i) data dependency and correlation among different types of sensors in heterogeneous WSNs are hard for SQL to capture; (ii) SQL is awkward in collaborative decision making and representing event triggering [7]; (iii) SQL does not explicitly support
probabilistic models; (iv) SQL-like languages do not naturally present a hierarchical model for event structure.

In contrast to SQL-like semantics, Petri nets are well accepted as a model to describe systems with distributed, concurrent, asynchronous, and non-deterministic features. Petri nets exhibit a number of properties that make them particularly useful for event description in WSN applications: (i) Extended Petri nets with a zero test (including Timed Petri nets and MEDAL) are equivalent to Turing Machines in computation power. (ii) Extended Petri nets can handle non-determinism and do not suffer from state explosion; (iii) Features inherited from Colored Petri nets allow us to incorporate spatial properties into the tokens, thus handling geographical constraints for WSN events; (iv) Features inherited from Stochastic Petri nets provide an advantage over other specification methods in terms of performance evaluation ability; (v) Petri nets have graphical support which is convenient for users who are not versed with logic or other complicated theoretical backgrounds.

Petri nets were first used to describe WSN events in [3] where a Sensor Network Event Description Language (SNEDL) was introduced. As a formal method, SNEDL is based on Petri nets, which allows it to rigorously and unambiguously specify WSN events. SNEDL aims to address key aspects of sensor networks such as temporal control, spatial constraints, heterogeneity, and probability issues. In SNEDL, a token is associated with type, capacity, time, and location attributes. A SNEDL representation of an event is given as a 10-tuple structure. We have developed a compact Event Description and Analysis Language (MEDAL) [4] which combines SNEDL’s expressive power with a more compact syntax while preserving SNEDL’s specification rigor. In addition, MEDAL models can be transformed into SNEDL and vice versa.

The MEDAL description of an event system in a sensor network can be given as a 7-tuple structure: $F = (P, T, A, \lambda, \beta, H, L)$ where $P$ is the set of places, $T$ is the set of transitions, $A$ is the set of arcs, $\lambda$ is the probability function for the arcs, $\beta$ is the temporal guard function, $H$ is the threshold function for places, and $L$ is the spatial guard function for transitions.

Sensors in MEDAL are abstracted by sensor events. Sensor events are denoted as places that take tokens from the environment. In Figure 1, places $T$, $A$, and $L$ are sensor events. Higher level events are constructed using sensor events. The number of sensor events is directly related to the types of sensors in the network. For example, if there are three types of sensors in the network - temperature, light, and acoustic sensors, then there are three types of sensor events in MEDAL: temperature, light, and sound. The tokens that arrive at each sensor event are associated with temporal and spatial attributes. Therefore, the information about when and where the data has been sensed can be retrieved. For example, if a token with a time stamp $t$, capacity $c$, and location attribute $(x, y, r)$ reaches a temperature sensor event (Figure 1), we can say that a temperature sensor at location $(x, y)$ with sensing range $r$ has detected temperature value $c$ at time $t$.

Sensor network events are a function of when and where they occur. MEDAL addresses the need of WSN applications for spatial and temporal semantics by incorporating both spatial and temporal logic.

**Temporal Logic:** Temporal logic refers to the temporal guard function $\beta$. It helps specify the temporal concepts “when” and “how long” in MEDAL Petri nets. $\beta$ guards all the transitions to ensure that they fire only during the specified temporal intervals. Introducing $\beta$ in MEDAL has practical importance because, in physical environments, some events can occur only during a particular temporal interval. For example, events that depend on sunshine can only take place during the day.

**Spatial Logic:** The geographic semantics of the application are enforced by the spatial function $L$. As a guard function for a transition $T$, $L$ ensures that the tokens carried by $T$’s incoming arcs satisfy the spatial locality conditions of the transition. $L$ is only applicable to higher-level events since we cannot constrain the sensing range of a sensor at the application level. If $L(T) = R$, the effective radius of an event should be equal to or smaller than $R$. In other words, there is a circle of radius $R$ associated with the node executing transition $T$. The locations of all incoming tokens should be within that radius in order for the transition to fire.

The current MEDAL logic provides basic support for expressing the spatial, temporal, and probability semantics of WSN applications. However, we believe that more complex WSN applications might not find this logic sufficient. To address this:

1. We are extending MEDAL to provide more complex marking-dependent probability functions. This will enable MEDAL to manipulate more complicated scenarios in sensor network applications.

2. We are developing event composition algebra in spatial, temporal, and probability dimensions with cross-dimension dependencies and interactions.

3. Since communication and power consumption are two very significant aspects of sensor networks, we are extending MEDAL to model both of them.

![Figure 1. MEDAL explosion detection model.](Image 339x644 to 515x720)
4. We are extending MEDAL to accommodate the specification of mobile network applications.

These modifications will greatly increase the analysis capabilities of MEDAL. They will allow MEDAL to model and analyze not only the logic of an application but also properties such as its communication model, remaining lifetime, and timeliness. By developing MEDAL models for different applications we are also investigating if further extensions are necessary in order to provide a language that is capable to fully model the properties of WSN applications.

An advantage of MEDAL is that it allows us to build event hierarchies where the nodes at a particular level perform only a subset of the application logic. Consider, for example, a scenario where an application uses three types of sensors to detect the presence and distinguish among three types of events: a person, a person with weapons, and a vehicle (Figure 2). Using MEDAL, the system designer develops a 3-tier classification approach to identify a particular event. The three tiers are node-level, group-level, and base-level event classification. At the node-level, transition $T_4$ processes readings from the sensors, checks necessary temporal and spatial conditions, and generates messages for the next level (group-level) processing. At the group-level, a cluster-head node collects readings from its neighboring sensing nodes. If the number of event reports is sufficient and satisfies the temporal and spatial conditions at transition $T_5$, the cluster-head forwards the information to the base station. At the third level, the base station runs classification algorithms to determine which event has occurred.

MEDAL’s structure allows us to divide the application model into hierarchical components. Each component (indicated as a dashed box) could be extended to a MEDAL sub-net to provide additional design details. Both transitions and places could be expanded out into sub-nets. This enables a more in-depth analysis of the application logic and provides a more thorough specification. We are designing techniques to help expand MEDAL models without compromising the preciseness, correctness, or clarity of the models.

To increase the efficiency and decrease the costs of event detection, applications that detect different events should be able to coexist on the same network of sensors. A next step would be to allow these applications to share common modules. Because of its Petri net structure, MEDAL allows for the MEDAL models of different applications to be superimposed. For example, if two applications share a common sub-event, their MEDAL models can be combined so that both applications can share the common sub-net instead of having their own copy. We are developing approaches to combine the MEDAL models of different applications running on the same network. Allowing models to share sub-events will help decrease the computational, memory, and communication requirements of event detection.

3. Specification transformation

Transforming the formal MEDAL model into executables that can be deployed on the sensor nodes is the next step an event service should perform. The process of recognizing the specified events is similar to a DNA transcript procedure. Therefore, the event recognition code generated from the MEDAL model and stored on the sensor nodes is called event-DNA. Each event-DNA is an encoded representation of a MEDAL model. Just like a MEDAL model, the event-DNA might represent the description of atomic or compound events.

The overall event service architecture we propose is shown in Figure 3. The MEDAL IDE (Integrated Development Environment) is an offline package which resides on a PC. Using this package, specified event semantics can be encoded and exported to a base node. The base node is responsible for the deployment of the event-DNAs onto the corresponding sensor nodes. For example, if an event-DNA represents a node-level event, it will be deployed onto every node, while if an event-DNA represents a group-level event, only a group leader will have a copy. The executables generated from the MEDAL model are stored in program memory. The event detection module achieves its goal by calling
other lower primitives such as reading sensor values, obtaining time and location information, as well as retrieving data from other program components.

The framework contains the following modules: (i) MEDAL environment, which includes both a specification module and an analysis module. It encodes the event specifications into event-DNAs and exports them to the base node; (ii) The Base node contains the encoded event-DNAs and has a deployment module in charge of transferring event-DNAs onto nodes; (iii) At the Sensor node level, the event detection module resides in program memory. The event detection module uses a token vector (called RNA) to communicate with other nodes during collaborative detection of higher level events.

Once the designers have the MEDAL model to describe their application, the next step is to write the code to be run on the nodes. A weakness of this step, however, is that manually translating the formal model into code might introduce bugs as well as lead to divergence of the code from the model. To address this, the event-DNA code could be automatically generated based on the MEDAL model of the application. An advantage MEDAL has over other event description approaches is that due to its formal structure and lack of ambiguity, it can be directly and automatically translated into code to be executed on the sensor nodes. Another advantage of MEDAL is that it is not tied in to any programming language. Therefore, depending on the platform, the generated code could be in a variety of languages such as nesC, Java, and C++, for example. This approach will significantly reduce the effort of writing code for the sensor nodes and will also improve code correctness.

More importantly, the generated code will follow the structure of the MEDAL model and the MEDAL logic objects, such as Places, Arcs, and Transitions, will be implemented explicitly. In accordance with MEDAL, the code execution is driven by tokens traversing through MEDAL objects. Code with such structure can 1) collect trace and other information; 2) differentiate between the execution of real events and any run-time tests that might be executed to verify the application’s correct behavior.

Currently we have an implementation of an automatic code generator that has allowed us to generate the code for a simple fire detection application [8]. The code generator takes the application’s Petri net model as input. The user is asked to provide this model by writing a script that specifies the details of the model such as places, transitions, how they are connected, and what sensors the sensor events correspond to. The code generator provides a small library containing predefined transition types which cover the majority of logical operations used in a WSN application. If, however, the user wants to specify transitions with more complex functionality, they are given the option to do so. Because of its MEDAL-based token flow programming structure, the generated code uses more memory than a manually written application. For comparison, for our fire detection application [8], the generated code, which is also augmented with run time testing support, is twice the size of the manually written application. However, since most of the code is automatically generated, the user only needs to provide the MEDAL specification script.

To evaluate our code generation mechanism we would design MEDAL models for a number of existing WSN applications and compare the properties of the generated code to those of the manually written one. We will compare parameters such as code size, memory requirement, correctness in terms of number of false positives and false negatives, and timeliness.

4. Conclusions

To satisfy the demands of the wide range of emerging WSN applications, sensor networks should provide sophisticated and robust event services. The goal of our work is to address this issue and develop event services that are well suited to describe and detect WSN events despite the numerous challenges such as limited resources, unreliable and imprecise sensor readings, and the constantly changing physical environment. We are designing a formal event specification language that will allow WSN designers to fully specify the events they are interested in. We are also working on facilitating and automating the translation from formal specification to code that can be executed by the nodes.

References