# Event-Based Systems for Detecting Real-World States with Sensor Networks: A Critical Analysis

Kay Römer and Friedemann Mattern
Institute for Pervasive Computing
ETH Zurich, Switzerland
{roemer,mattern}@inf.ethz.ch

**Abstract**

*Wireless sensor networks can be considered as a tool for detecting certain states in the real world. We examine the use of event-based approaches for this task. In the literature, a number of event notification systems have been presented that facilitate the specification and automatic detection of event patterns – so-called composite events. While events are a valuable abstraction in sensor networks, we show that composite events are less suited to detect real-world states with sensor networks. We illustrate an alternate solution that retains the advantages of an event-based approach, but which provides better support for the specification and detection of real-world states.*

## 1. Introduction

Sensor Networks can be considered as a tool for observing states of the real world. For example, the occupancy of nesting burrows is observed in [23], or the temperature distribution across a vineyard is observed in [2]. In particular, we are often interested in the detection of predicates over the state of the real world, such as "Notify me if more than four birds are in their burrows" or "Notify me if the temperature in a given area is below 0°C". We refer to such problems as *detecting a certain real-world state*.

The detection of such real-world states is a non-trivial problem due to the distributed nature of sensor networks. Using sensors, nodes collect state samples, where each sample represents a rather local view of the real world at a certain time and place. In order to obtain a more global and consistent estimate of the state of the real world, many such local samples have to be integrated in a process known as data fusion or data aggregation.

Data fusion involves communication of the local state samples in order to aggregate two or more samples from distinct nodes. However, limited resources in sensor networks make it a necessity to reduce the communication overhead to a minimum. Hence, rather than transmitting local state information in regular intervals, only indications of changes in the observed local state are usually transmitted. Such state changes are commonly referred to as *events*. Hence, state detection is closely related to the notion of events.

Events proved to be a useful abstraction in many traditional application domains such as database systems and distributed systems. In the context of distributed systems, various event notification systems (e.g., [4], [5], [21]) have been devised that assign network nodes the roles of event *producers* and event *consumers*. Event producers generate typed event notifications upon the occurrence of certain local events. Event consumers specify their interest in certain types of event notifications using so-called *subscriptions*. An event notification service mediates between producers and consumers by providing efficient delivery of event notifications from producers to consumers based on subscriptions.

An important class of event systems supports *content-based* subscriptions, which do not only specify certain types of events, but can also specify certain instances of an event type by means of expressions over event parameters. For example, an event may contain the location of the sensor node. A content-based subscription may then declare interest in events from sensor nodes that are located within a certain area. Some of these distributed event systems support the notion of *composite events*, where the consumer specifies a pattern of multiple events. The event notification system matches event notifications against these patterns and informs the consumer when a match is found [6], [10], [12], [15].

Such content-based event systems with support for composite-event detection are a potential tool for supporting the detection of real-world states in sensor networks, since they support the communication of local state changes (by means of event notifications) and the detection of global real-world states (by means of composite-event detection).

The purpose of this paper is to examine the applicability of such event notification systems for sensor networks. In Section 2 we show that existing systems for the detection of composite events are not particularly well-suited for detecting complex real-world states. In Section 3 we discuss various issues that have to be considered when designing systems for state detection with sensor networks. In Section 4 we present an initial prototype system for state detection in sensor networks that addresses some of the raised issues. Since much of our discussion is closely related to existing work on event-based systems, we point the reader to related work where appropriate, rather than discussing related work separately.

## 2. States vs. Events

Although we are interested in detecting certain partial states of the real world, the notion of events is relevant to our problem,

since events can be considered as state transitions. Given an initial real-world state, certain sequences of events may lead to a target state that is to be detected. Hence, the detection of interesting real-world states is closely related to the detection of event patterns.

*Event notifications* can be considered as software representations or real-world events. An event notification typically consists of a type specification and of an arbitrary number of parameters, where each parameter has a name, a type, and a value.

Events are a particularly useful abstraction in the context of sensor networks because of several reasons. Firstly, events provide an implicit data compression, since only changes in the state of the real world lead to the generation and communication of an event notification. Secondly, the occurrence of a physical event and the resulting delivery of an event notification triggers computation to evaluate or to react to the change in the real world. That is, if the relevant aspect of the real world doesn't change, neither communication nor computation is triggered. Hence, events are a natural way to optimize the consumption of communication and computation resources, which is an important issue in resource-constrained sensor networks.

A number of research projects are concerned with the detection of *composite events* – sets of events that fulfill certain constraints on their types and parameters. The user specifies a composite event using a declarative language. The event notification system parses these specifications and tries to match them with actual event notifications. Whenever a match is detected, the system generates a new event notification representing the composite event.

Different systems use different kinds of event representations and provide different sets of predicates for specifying composite events. Basic predicates provided by almost all systems are the occurrence of an event (e.g., A happened), temporal predicates (e.g., A happened *before* B (A $\rightarrow$ B), A and B happened *within* X seconds), and predicates over event parameters (e.g., A.temperature - B.temperature $< 5$). Logical operators can be used to formulate compound predicates over other predicates.

Let us consider the question whether such systems for composite event detection are a suitable tool to support the detection of real-world states with sensor networks. To answer this question, we assume a simple model where each sensor node can assume states $\{s_i\}$ depending on the output value of the sensors attached to it. In this model, each node operates independently of all other nodes. Based on these states, a finite automaton can be constructed that specifies possible state transitions. Let us assume that the state transitions in this automaton are mapped to event notifications that contain a node identifier as a parameter. That is, whenever the sensor reading on a node leads to a state transition, a corresponding event notification is emitted.

As an example, let us consider a sensor node that can detect the proximity of a physical object. The corresponding automaton would then contain two states $s_1$ ("object present") and $s_2$ ("object not present") with two state transitions and corresponding events detect($i$) and lost($i$), where $i$ is the identifier of the node. The automaton is depicted in Figure 1 (a), where the state names are abbreviations for "present" and "not present", and where the transition labels are abbreviations for "detect($i$)" and "lost($i$)".

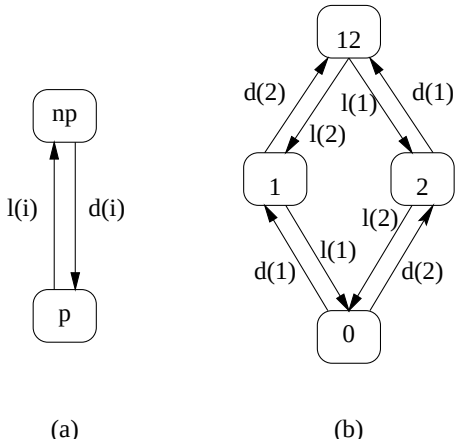


**Fig. 1:** (a) A node automaton represents the possible states and transitions of a single sensor node. (b) The observation automaton represents real-world states based on the individual states of many sensor nodes.

Based on these local views of individual sensor nodes, more complex (i.e., composite) real-world states can be specified that involve multiple sensor nodes. For this, another set of states $\{os_i\}$ are used that describe certain real-world states that can be observed by two or more sensor nodes. Accordingly, a state automaton can be defined based on these states.

Continuing with the above example, we want to detect the situation where $N$ (close-by) sensor nodes detect the object concurrently. This is a common task in sensor networks to reduce the probability of false observation results. The automaton for $N = 2$ would then contain four states $os_1$ ("object not present"), $os_2$ ("object present at node 1"), $os_3$ ("object present at node 2"), and $os_4$ ("object present at nodes 1 and 2"). The automaton is depicted in Figure 1 (b), where the state names indicate the nodes that currently see the object. Transitions are labeled with the event notifications that trigger the respective state transition.

Detecting a real-world state by means of composite event detection can now be described as follows. Given an initial state (e.g., "0" in the example) in the observation automaton and a target state that is to be detected (e.g., "12" in the example), detect all event sequences that imply a state transition from the initial state to the target state. Such matching event sequences can be considered as composite events. The question now is, whether systems for composite-event detection are adequate tools for describing and detecting these composite events.

The answer to this question inherently depends on the complexity of the observation automaton. In general, if there is only a single or few matching event sequences, these can often be easily described as composite events. However, in many realistic observation automata, there will be many different

matching event sequences. In many cases, it is not possible to capture all these with a single compact composite event declaration. In the above example, there is an infinite number of matching event sequences, since various loops of state transitions can be constructed. For example:

- $d(1) \rightarrow d(2)$
- $d(1) \rightarrow l(1) \rightarrow d(2) \rightarrow d(1)$
- $d(1) \rightarrow l(1) \rightarrow d(2) \rightarrow l(2) \rightarrow d(1) \rightarrow d(2)$

With many existing composite-event systems, it is not possible to specify a compact and easily comprehensible composite-event declaration that captures all matching event sequences (in particular for $N > 2$). We conclude that even for simple state specifications as in our example, the corresponding composite-event-detection problem may not be efficiently describable.

One potential approach to address this problem is to allow a more direct specification of the interesting real-world states, rather than having the user manually translate state descriptions into complex composite-event descriptions. In the above example, such a specification could read "two nodes are in state *p* concurrently".

In [9], a composite-event detection system for sensor networks is described that goes one step into this direction. By associating a *validity time interval* with each event notification, an event can in fact be considered as a specification of the current state of a sensor node. Assuming there are two states $s_1$ (default state) and $s_2$ in the node automaton, the node changes from $s_1$ to $s_2$ at the occurrence of the event and returns to $s_1$ after the validity interval has elapsed.

With this system, a real-world state can then be specified by requesting that one or more events with overlapping validity intervals exist. This is effectively equivalent to requesting that certain nodes are in specific states concurrently. The use of the system is illustrated in [9] by an example quite similar to ours: an explosion is to be detected, where nodes are equipped with temperature, acoustic, and light sensors to detect excessive heat, a bang, and a light flash. Upon detection of these events, respective event notifications with fixed and predetermined validity intervals are generated. An explosion is detected if these events occur with overlapping validity intervals.

Note that fixed validity intervals are a simplification that is not appropriate in many situations. For example, it cannot be known in advance how long the heat will last in reality. In our earlier example, it is usually not known in advance when exactly the monitored object will leave after it has been detected by a sensor node.

Yet another approach would be to specify a target state by means of the observation automaton itself. However, observation automata can become quite large and complex if many sensor nodes are involved. In the worst case, the number of states grows exponentially with the number of sensor nodes involved. Hence, this approach would be rather clumsy. Moreover, only few states of the automaton might be of interest to the application, such that large parts of the automaton may not be significant to the problem.

In summary, we seek state specifications that are compact and readable, but allow the specification of a range of complex states.

## 3. System Issues with State Detection

As discussed in the previous section, events are a useful abstraction to specify and deliver notifications about state changes at individual sensor nodes. However, for the specification of complex states involving multiple sensor nodes, events and in particular composite events are less suited. Hence, we want to consider systems that use events as a basic abstraction to deliver information about the states of individual nodes, but where complex real-world states (involving multiple sensor nodes) can be specified more efficiently and conveniently compared to traditional approaches for composite-event detection.

Such systems have to provide a number of functional components and have to deal with various challenges posed by sensor networks. We will briefly discuss these issues in the following subsections.

### A. State and Action Specifications

State specifications declare (abstractions of) real-world states that are of interest to a sensor network application. The declaration of such a state is based on the individual states of a set of sensor nodes. Various predicates allow the specification of constraints on the actual states of the involved sensor nodes.

Each such state specification is accompanied by an action (cf. Event-Condition-Action (ECA) rules [16]) that is executed whenever the actual configuration of the sensor network matches the respective state specification. Action specifications can take a number of different forms, such as a code snippet in a specific programming language, or the specification of an event notification that should be generated when a match occurs. A pair of a state specification and an action is called a rule. A system for state detection typically accepts a list of such rules.

### B. State Detection Algorithm

A state detection algorithm is an online algorithm with two inputs: a list of rules and the current state configuration of the sensor nodes. As discussed before, the algorithm can be informed of the current states of the sensor nodes by means of event notifications that represent state transitions at the sensor nodes. The algorithm should detect matches as soon as sufficient information is available.

State detection can be performed on sensor nodes or outside of the sensor network on a dedicated, more powerful computer system. If performed on sensor nodes, the detection algorithm must meet significant constraints on computation and storage resources, since typical sensor node hardware provides only few kilobytes of memory and few MIPS. Hence, state specifications should be designed such that efficient matching algorithms are feasible, which can impose constraints on the complexity of state specifications.

### C. Distributed Detectors

In sensor networks, communication is a rather scarce resource, since every transmitted bit of data consumes a certain amount of the constrained energy budget of a sensor node. Hence, rather than transmitting large amounts of data through the network, raw data should be processed (and compressed) as close to its source as possible. In other words, a subset of sensor nodes should collect and process data from other, nearby nodes in order to save energy and communication bandwidth [7], [11]. If state matching should be performed inside the sensor network, rather than at a centralized computing system outside of the sensor network, the limitations on the matching algorithm sketched in the previous section are of practical relevance.

This immediately raises the question which nodes should execute the matching algorithm in order to optimize certain parameters such as energy consumption [3]. The remaining nodes must then send event notifications to one or more of these *detector nodes*. In some cases, the selection of detector nodes may change dynamically at runtime in order to adapt to real-world dynamics (e.g., mobility of an observed object).

A necessary precondition for distributed detector nodes is that the global predicate that is to be detected can be decomposed into local predicates, such that each local predicate considers only the states of a few sensor nodes. These local predicates can then be detected by distributed detector nodes. However, there may exist global predicates with practical relevance that cannot be decomposed in this way. For example, the specification "Notify me if two temperature sensors observe a difference ¿ 5°C" requires that a detector node has access to the sensor readings of all nodes. For such predicates, state detection must be performed by a single instance of the detection algorithm that knows the individual states of all sensor nodes.

### D. Time Synchronization and Node Localization

As illustrated by the example in Section 2, temporal and spatial relationships among sensor nodes are a key element for combining events generated by different sensor nodes. State specifications will therefore typically include spatial and temporal predicates for expressing real-world states in terms of spatio-temporal relationships among the states of a set of sensor nodes. Support for these predicates requires time synchronization among sensor nodes and localization of sensor nodes in physical space.

Temporal relationships also play an important role in traditional distributed systems. However, for many applications the detection of causal or "happened before" relationships is sufficient. Logical time [8], [13] can provide an efficient implementation for detecting these relationships. However, logical time only captures relationships between "in system" events, defined by message exchanges between network nodes. In contrast, events in sensor networks are triggered by real-world phenomena. Hence, events that are detected on different sensor nodes would usually be causally unrelated in the above sense. Therefore, physical time must be used to relate events in the physical world.

The precision of time synchronization and localization is critical for the correctness of state detection, since even small errors can lead to wrong results. In our example, the event sequence $d(1) \rightarrow l(1) \rightarrow d(2)$ does not lead to the detection of the desired target state "two nodes see the object concurrently". However, if due to synchronization errors $l(1).t > d(2).t$ although $l(1)$ happened before $d(2)$ in reality, then the system will observe $d(1) \rightarrow d(2)$, which results in a match. One potential approach to this problem is the use of intervals instead of point estimates to represent time and location. However, this may lead to situations where temporal and spatial predicates cannot be decided. For example, if time is represented by intervals, then the predicate $d(1) \rightarrow l(1)$ cannot be decided if the intervals for $d(1).t$ and $l(1).t$ overlap [17].

### E. Event Ordering

Related to the issue of time synchronization is the problem of temporal ordering of events. If event notifications are tagged with a time stamp according to a globally synchronized time frame, a receiver can order events from different senders. However, the receiver cannot easily decide at some time $t$ whether all events with a time stamp earlier than $t$ have arrived yet. This is an important problem, since a "missing" event can falsify the result of state detection. Consider again our example and the event sequence $d(1) \rightarrow l(1) \rightarrow d(2)$, which does not lead to a match. However, if $l(1)$ is delayed in the network and arrives after $d(2)$, the resulting event sequence $d(1) \rightarrow d(2)$ will lead to a match.

This problem is particularly relevant in sensor networks, where message delivery is subject to long, variable, and unpredictable delays. Hence, it is rather likely that messages will arrive out of order at a receiver.

In conjunction with logical time, causal ordering algorithms can be used to ensure that causally related events are delivered to a receiver in causal order [14], [20]. However, as mentioned in the previous section, logical time is usually not sufficient in sensor networks. Therefore, causal ordering algorithms cannot be applied and event ordering must be based on physical time. Potential approaches to this problem in the context of sensor networks are discussed in [18].

### F. Event Delivery

Many classical event-notification systems have been designed for networks with a static topology. However, node additions, removal, and mobility lead to dynamic topologies and may cause temporary partitions in sensor networks. Event delivery has to deal with these dynamic effects that are typical for sensor networks. Hence, event routing algorithms have to be revisited for dynamic sensor networks [5].

Moreover, event delivery must deal with temporary disconnects in the network [22]. This raises the questions how to deal with events that cannot be forwarded to the receiver due to temporary network partitioning. There are basically two

options to deal with this problem: store the events until the network becomes re-connected, or drop events. The first option is problematic due to the limited storage capacity of typical sensor nodes. The second option is also problematic, since lost events may falsify observation results similar to delayed events as discussed in the previous section.

## 4. A Prototype System for State Detection

In this section we present a prototypical system for state detection that addresses some of the issues raised in the previous section.

### A. State and Action Specifications

As motivated in Section 2, we assume that individual sensor nodes emit event notifications whenever they make a transition in their local state automaton. Specifications of complex states are based on these events arriving form one or more sensor nodes. Such an event $ev(id, t, l, a_1, ..., a_k)$ has type $ev$ and contains a number of parameters: $id$ is a unique identifier of the node that generated the event, $t$ and $l$ are time and location of the sensor node when the event was generated. Optionally, any number of additional parameters $a_i$ can be contained in an event notification. We will refer to the value of parameter $x$ of event $e$ as $e.x$. Our system allows to reconstruct node states from event notifications by means of the *state* operator, which comes in two variants:

- $\mathrm{state}(e, t, P)$, where $e$ is a placeholder for an event, $t$ a duration, and $P$ a predicate over the parameters of $e$. If $P$ is omitted, it is assumed to be "true". The specified state is entered at time $e.t$, where the placeholder $e$ is matched by an actual event notification for which $P$ evaluates to true. The state is left again at time $e.t + t$. Note that this variant is similar to validity intervals introduced in [9].
- $\mathrm{state}(e_1, e_2, P_1, P_2)$, where $e_i$ are event placeholders and $P_i$ are predicates. $P_1$ can refer to the parameters of $e_1$, $P_2$ can refer to parameters of $e_1$ and $e_2$. If one or both expressions are omitted, they are assumed to be "true". The specified state is entered at $e_1.t$ where the placeholder $e_1$ is matched by an actual event for which $P_1$ evaluates to true. The state is left at time $e_2.t$ if $e_2$ is matched with the earliest event after $e_1$ for which $P_2$ evaluates to true.

Events and states can be assigned a name $n$ by the statement $n : x$, where $x$ is a state or event specification. Consider the example in Section 2. We can specify the state "present" in the node automaton by the statement

```
present : state(d, l, true, d.id == l.id)
```

Our system provides a number of predicates to formulate more complex state specifications. These predicates fall into three larger classes: temporal predicates, spatial predicates, and predicates over event parameters. Logical operators can be used to formulate compound statements over these predicates. Temporal predicates are used to express temporal relationships among different node states. A node state can be considered as a time interval that begins when the state is entered and that ends when the state is left. However, after a state has been entered, it is not known when the state will be left in the future, resulting in intervals with open ends. We say that a state is *active* during this time interval. Based on this observation, a number of temporal predicates on states (i.e., open-ended intervals) can be defined.

- $\mathrm{overlap}(s_1, ..., s_k)$ evaluates to true if there is a point in time where all states $s_i$ are active.
- $\mathrm{disjoint}(s_1, ..., s_k)$ evaluates to true if the states $s_i$ have all been active, but there is no point in time where two or more states have been active concurrently.
- $\mathrm{contain}(s_1, ..., s_k)$ evaluates to true if state $s_i$ is active, but only while $s_{i-1}$ is active.
- $\mathrm{after}(s_1, ..., s_k)$ evaluates to true if $s_i$ is activated only after $s_{i-1}$ has been deactivated.
- $\mathrm{afterbeg}(s_1, ..., s_k)$ evaluates to true if $s_i$ is activated after $s_{i-1}$ has been activated.
- $\mathrm{chain}(s_1, ..., s_k)$ evaluates to true if $s_i$ is activated while $s_{i-1}$ is active, and if $s_i$ is deactivated after $s_{i-1}$ has been deactivated.
- $\mathrm{within}(t, e_1, ..., e_k)$ evaluates to true if all events $e_i$ occurred within $t$ seconds. Usually, the $e_i$ refer to events in the state specifications in order to implement time limits.

Note that these predicates are conceptually similar to interval arithmetic [1], [24]. However, with open-ended intervals, our system has to deal with incomplete knowledge about interval ends, since a match should be reported by the state detection algorithm as soon as sufficient knowledge is available to make a decision.

While a state defines a region in time based on the time of occurrence of events, a similar approach can be applied to define regions in space based on the location of events. In our system, space regions are described by spheres:

- $\mathrm{region}(e_1, ..., e_k)$ is the smallest possible sphere that contains all $e_i.l$.
- $\mathrm{region}(e_1, r_1, ..., e_k, r_k)$ is the smallest possible sphere that contains all the spheres $(e_i.l, r_i)$ (i.e., a sphere with radius $r_i$ and center point $e_i.l$).

In analogy to temporal predicates over states, our system supports spatial predicates (i.e., overlap, disjoint, contains, within) over space regions.

A state specification then consists of a logical statement over the above predicates. Let us consider the example from Section 2 again. We can specify the target state as follows using our system:

```
overlap(state(n1 : d, l, true, d.id == l.id),
        state(n2 : d, l, true, d.id == l.id))
AND n1.id != n2.id
```

Here, two instances of the node state "present" are defined as explained above. The activation event `d` of the two state instances is bound to the names `n1` and `n2`, respectively. Then, we use the `overlap` predicate to specify a target state where two nodes detect the object concurrently. In order to ensure that both states refer to different nodes, we request that the `id`'s of the activation events are different.

As noted earlier, each state specification is accompanied by an action. In our system, the action is a snippet of C code. The action code can access all the events that led to the state match. Additionally, a number of special functions are provided that can be invoked from the action code. These can be used to generate new events and to control the state detection algorithm as will be described in the following section.

Each rule (i.e., pair of a state specification and an action) consumes input events and potentially produces new events. The system allows to connect these rules, such that the output events of one rule serve as input events to another rule. This feature can, for example, be used to specify node states with more complex activation conditions. In general, a rule specification has the following format:

```
rule <name> from <input rules> state <state spec>
  { <action code> }
```

where `name` is an identifier for the rule, `input rules` is a list of rule names from which to take input events, `state spec` is the state specification, and `action` is C code that defines the action.

### B. State Detection Algorithm

The state detection algorithm matches events against a set of rules. Each incoming event is copied for each rule and rules are then handled independently of each other.

The detection algorithm then tries to find an assignment of actual event notifications received from sensor nodes to the placeholders in the state specification. A single event may be used to match different placeholders. If multiple assignments would result in a match, then the one with the oldest events is selected. By default, events that participated in a match are consumed so that they cannot be used in subsequent matches. However, as part of the action code, it may be requested to reuse some events for future matches. Multiple rules can match using the same set of event notifications.

Our current matching algorithm is based on backtracking. In each step of the algorithm, an unmatched event placeholder in the state specification is assigned an event, such that the conditions of the state specification are not violated. If such an assignment is not possible, the algorithm backtracks to revise an earlier assignment. This proof-of-concept algorithm is optimized for memory consumption, but its worst-case runtime is exponential in the number of event notifications.

### C. Further Issues

Our prototype system currently uses the sensor-network setup described in [19]. The detection algorithm is executed on a computer system outside of the sensor network (i.e., we did not address the problem of detector placement so far). We also assume that the network is always connected (i.e., network partitions are not addressed so far).

The time synchronization algorithm [17], [19] leaves the clocks of the sensor node unsynchronized, such that each node has a different local time-scale. Time stamps (referring to the time-scale of the sender) are transformed to the time-scale of the receiver as the event is forwarded hop-by-hop through the network. The algorithm uses intervals to represent time.

The message ordering algorithm [19] is based on the knowledge of a maximum network latency $\Delta$. Delaying the evaluation of inbound events for $\Delta$ will ensure that out-of-order messages will arrive during this artificial delay and can be ordered correctly using their time stamps. That is, message ordering can be achieved without additional message exchanges. By measuring the delay of events (which is possible due to time synchronization), we maintain a running estimate of $\Delta$. With this approach, it is possible though unlikely that events are delivered in the wrong order. In our current prototype we deal with this problem by notifying the application of such exceptional cases.

## 5. Conclusion

In this paper, we considered the problem of detecting real-world states with sensor networks using event-based approaches. We showed that events are an appropriate abstraction in sensor networks, but composite-event detection is ill-suited to support the efficient specification and detection of complex real-world states. We discussed various issues that have to be considered when designing systems for state detection with sensor networks. We presented a prototype system that addresses some of these issues.

There is much room for future work. Some of the issues discussed in Section 3 are not yet addressed by our prototype system, others are only partially addressed (see Section 4-C). Some of these open issues have been addressed by other researchers in different contexts, but it remains to examine whether these approaches can be applied to the sensor-network domain. One important issue is the improvement of the state detection algorithm so as to allow its execution on sensor nodes.

## References


[1] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] R. Beckwith, D. Teibel, and P. Bowen. Pervasive Computing and Proactive Agriculture. In *Adjunct Proc. PERVASIVE 2004*, Vienna, Austria, April 2004.

[3] B. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *IPSN*, Berkeley, USA, April 2003.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service. In *Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, Portland, OR, July 2000.

[5] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Middleware*, Rio de Janeiro, Brazil, June 2003.

[6] R. Hayton. *OASIS: An Open Architecture for Secure Interworking Services*. PhD thesis, University of Cambridge, 1996.

[7] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *6th Intl. Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, USA, August 2000.

[8] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(4):558–565, July 1978.

[9] S. Li, S. H. Son, and J. A. Stankovic. Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. In *IPSN 2003*, Palo Alto, USA, April 2003.

[10] C. Liebig, M. Cilia, and A. Buchmann. Event composition in time-dependent distributed systems. In *CoopIS*, Edinburgh, UK, September 1999.

[11] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *OSDI 2002*, Boston, USA, December 2002.

[12] M. Mansouri-Samani and M. Sloman. GEM – A Generalised Event Monitoring Language for Distributed Systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4(25), February 1997.

[13] F. Mattern. Virtual Time and Global States in Distributed Systems. In *Workshop on Parallel and Distributed Algorithms*, Chateau de Bonas, October 1988.

[14] F. Mattern and R. Schwarz. Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail. *Distributed Computing*, 7(3):149–174, 1994.

[15] G. J. Nelson. *Context-Aware and Location Systems*. PhD thesis, University of Cambridge, 1998.

[16] N. Paton. *Active Rules in Database Systems*. 1999.

[17] K. Römer. Time Synchronization in Ad Hoc Networks. In *MobiHoc 2001*, Long Beach, USA, October 2001.

[18] K. Römer. Temporal message ordering in wireless sensor networks. In *IFIP Mediterranean Workshop on Ad-Hoc Networks*, pages 131–142, Madhia, Tunisia, June 2003.

[19] K. Römer. Tracking Real-World Phenomena with Smart Dust. In *EWSN 2004*, Berlin, Germany, January 2004.

[20] A. Schiper, J. Eggli, and A. Sandoz. A New Algorithm to Implement Causal Ordering. In *Workshop on Distributed Algorithms*, pages 219–232, Nice, France, 1989.

[21] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *AUUG 97*, Brisbane, Australia, September 1997.

[22] P. Sutton, R. Arkins, and B. Segall. Supporting Disconnectedness – Transparent Information Delivery for Mobile and Invisible Computing. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 01)*, Brisbane, Australia, May 2001.

[23] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a Sensor Network Expedition. In *EWSN 2004*, Berlin, Germany, January 2004.

[24] T. Wahl and K. Rothermel. Representing time in multimedia systems. In *Multimedia Computing and Systems*, Boston, USA, May 1994.