International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 2, Number 1, pages 2–17, January 2012

Execution and Time Models for Pervasive Sensor Networks

Ajay D. Kshemkalyani
Department of Computer Science, University of Illinois at Chicago
Chicago, IL 60607, USA

Ashfaq A. Khokhar
Department of Electrical and Computer Engineering, University of Illinois at Chicago
Chicago, IL 60607, USA

Min Shen
Department of Computer Science, University of Illinois at Chicago
Chicago, IL 60607, USA

Received: July 25, 2011
Revised: October 31, 2011
Accepted: December 15, 2011
Communicated by Akihiro Fujiwara

Abstract

Sensor-actuator networks and interactive ubiquitous environments are distributed systems in which the sensor-actuators communicate with each other by message-passing. This paper makes three contributions. First, it gives a general system and execution model for such sensor-actuator networks in pervasive environments. Second, it examines the range of time models that are useful for specifying properties, and for implementation, in such distributed networks, and places approaches and limitations in perspective. Third, it shows that although the partial order time model has not been seen to be useful as a specification tool in real applications of sensornets, yet, it is useful for real applications in pervasive sensornets because (under certain conditions) it can serve as a viable alternative to physically synchronized clocks that provide the linear order time model.

Keywords: pervasive networks, sensor networks, time models

1 Introduction

A pervasive environment can be thought of as a collection of networked autonomous embedded systems that interact with the physical world through sensors and actuators. Such systems aim to sense-monitor-control the physical world. The monitoring is achieved via tracking a time-dependent map or mirror of the spatio-temporal activities in the physical world [19]. The system can be thought of as a "bridge to the physical world" [11] or its most approximate instrumentation [12].

In the sensing and monitoring phases, a central issue is that of creating and monitoring the time-varying global map of the physical world, and evaluating predicates on that map. In the most general case, the predicate is on a pattern of events in the map and has two components – a spatial component and a temporal component, on the monitored variables [6]. The temporal component specifies various timing relations on the observed values of the variables/system attributes that need to be satisfied by the predicate. The most common of these is the "instantaneous" snapshot of the variables, for example, assumed in pervasive systems [4, 5, 16, 30, 36, 37]. More complex timing relations exist, and can be specified using logical time or physical time or a combination of both. In fact, there are many temporal algebras and temporal logics (*TL*) for reasoning with distributed system executions, that have been proposed over the past three decades and which can be used to specify the timing relations. Several have also been proposed for sensor networks and pervasive environments; see any recent paper such as [6] for a survey of these frameworks. But at the core, these diverse specifications assume either (i) a *single time axis model or the total order model*, or (ii) a *multiple time axis model or the partial order model.* Assembling snapshots satisfying various timing specifications under these two time models has been well-studied in the distributed computing literature [23]. However, this has not received enough attention in sensornets, with their features such as limited energy sources, mobility patterns, and the inability to trace communication in the physical world. Further, pervasive environments use sensed properties to determine context and adapt their behavior. The sensed properties are application dependent and the application's demands are different from those in traditional distributed systems.

The two time models - the single time axis model and the multiple time axis model - have several uses in the network layer, operating system layer, middleware layer, and application layer in traditional systems. These uses are outlined in Appendix A.

In this paper, we make the following contributions.

1. We give a general system model and execution model for sensornets in pervasive environments in Section 2.

2. We chart out a time model space for specifying properties to be detected in Section 3.1, and a time model space for realizing (implementing) the specifications is described in Section 3.2. Section 3 also serves as a survey of time models, covering the background and related works.

3. We examine the applicability of the single time axis and the multiple time axis models for pervasive environments containing embedded sensors/actuators, and place approaches and limitations in perspective in Section 4.

   We conclude that presently, the partial order time model for specification of predicates has not found adequate uses in pervasive sensor-actuator networks. Rather, the single axis time model continues to be widely used. On the other hand, we also explore the options of implementing the single axis time model. While physical clock synchronization protocols are clearly a desirable option to provide the single time axis, we make a case that that in some applications (characterized by the unavailability and or the high cost of such clock synchronization protocols), logical time strobe clocks [25] that provide a partial order of time are a viable alternative.

Section 5 gives some application scenarios where the proposed time model can be used. Section 6 gives a concluding discussion and lists open problems.

2 System and Execution Model

2.1 System Model

Sensor-actuator networks and pervasive environments are distributed systems that interact with the physical world in a sense-and-respond manner. A primary function is sensing the activities in the physical world and responding to them; thus, monitoring and collecting the global state, evaluating it for some time-related predicates, and responding back to the environment forms a generic loop. Pervasive systems additionally use the sensed properties to determine context and adapt their behavior.

A sensor-actuator network or a pervasive environment can be modeled at the application layer as a quadruple $\langle P, L, O, C\rangle$, where:

- P is a set of sensor/actuator processes which have access to some form of clock,
- L is a logical network overlay over which the processes in P can communicate with each other in an asynchronous message-passing manner,
- O is a set of external world objects, each with a set of attributes, that can be sensed and/or controlled by the sensor/actuator processes, and
- C is a logical network overlay in the physical world over which the objects in O communicate (in a synchronous or asynchronous manner).

$\langle P, L \rangle$ forms the *network plane* or the observation-and-control plane. The processes in P may be static or mobile (e.g., hand-held sensing devices or robots) and may communicate over wired or wireless media with one another over L. A process in P can also sense and actuate the objects in its range. (In a common configuration, a distinguished process P_0 acts as a root or back-end server that processes the sensed information.) L is a dynamically changing graph.

$\langle O, C \rangle$ forms the *world or physical plane.* The objects in O may be static or mobile (e.g., objects with RFID tags, animals with embedded chips, humans). These objects can be sensed by and/or can receive actuator signals from processes in P, but have no independent access to a synchronized clock. The objects in O can communicate with one another over the physical world overlay C; such communication may or may not be sensed by the processes in P and hence may not be replicable in L. Such channels in C have been termed as covert or hidden channels [2, 18]. C is also a dynamically changing graph.

Distinguishing features of $p \in P$ versus $o \in O$ are:

1. p is an active network entity whereas o is not;
2. p has access to an independent clock whereas o does not;
3. p (usually) exhibits deterministic behavior whereas o as part of the real-world need not behave deterministically or predictably.

In some cases, an entity may play a dual role of p and o, as in a zebra with an embedded sensor in a zoo, or a smart pen in an intelligent office.

2.2 Execution Model

A traditional message-passing distributed execution operates on the "network plane". To adapt the network plane model $\langle P, L \rangle$ to sensor/actuator networks, we enhance the standard model of an asynchronous message-passing distributed execution (see [23]) as follows. At each process $P_i \in P$, the local execution is a sequence of alternating states and state transitions caused by events. An event e is one of three types:

- An internal event, which is of type: *compute* (c), *sense* (n), or *actuate* (a). Although the n and a types of events are communication events, this communication is between the passive environment object in O (which does not have access to a synchronized clock) and the active process(es) in the sensor/actuator network, which has access to some form of synchronized clock.
- A *send* event (s), at which a message is sent by a process in P to another process in P. The sent message is timestamped by the sender's clock value (whether physical or logical, scalar or vector). A send event is semantically determined by the program logic. Note that if a communication send event over a covert channel between two objects in the $\langle O, C \rangle$ plane can be detected (which current technology cannot), the event can also be mirrored in $\langle P, L \rangle$.
- A *receive* event (r), at which a message is received by a process in P from another process in P. The received message is timestamped by the sender's clock value, and standard rules from the distributed computing literature [13, 26, 27] can be used for updating the receiver's clock.

A receive event is semantically determined by the program logic. If a communication receive event over a covert channel at an object in the $\langle O, C\rangle$ plane can be detected (which current technology cannot), the event can also be mirrored in P.

Whenever a significant change in the value of an attribute of an object is sensed by a sensor/actuator process, it records a sense event n. A message send event s is triggered at a sensor/actuator process to communicate information about a relevant sensed event to other sensor/actuator processes (or as a special case, to a distinguished root process P_0) so as to enable on-line detection of a global predicate on the attributes of sensed values of objects across the system. If the predicate is satisfied, a message send event is also triggered to actuate one or multiple sensor/actuator nodes to output to the environment objects. It is important to note, however, that the two types of send events described above and also the corresponding receive events belong to the *control* computation that monitors the state of the system. They cannot track true causality of the world plane. These control messages are inherently artificial, but they implement cause-and-effect in the system by the interaction of the $\langle O, C\rangle$ plane with the $\langle P, L\rangle$ plane.

Variables are of two kinds: each object $o_i \in O$ has an attribute $o_i.a$ (this can be generalized to multiple attributes per object). This variable can be tracked by multiple sensors/ actuators. In addition, each sensor/ actuator process $p_i \in P$ has local variables to track object attributes and maintain state.

We have used an event-driven execution model. An event occurs whenever a monitored value, whether discrete or continuous, changes significantly. The time duration between two successive events at a process identifies an interval. We model the event-driven activity at processes in terms of intervals. The application seeks to detect a predicate ϕ that is (i) explicitly defined on attribute values during intervals, that are (ii) implicitly related using certain timing relationships. The most popular timing relationship, "concurrent" among the intervals, captures the notion of *Instantaneously* or the instantaneous observation of the physical environment. However, note that more complex timing relationships are possible, as discussed in Section 3.

3 Time Model for SensorNets

Traditional asynchronous message-passing distributed systems use two common time models [23]: (i) the single time axis model that loosely corresponds to the event interleaving model of the distributed execution, and (ii) the vector time model, that loosely corresponds to the partial order model of events in the distributed execution.

To better understand the design space for specification of timing properties, and the design space for implementing timing mechanisms, specific to sensor/actuator networks, we identify these two design spaces next. The specification design space is driven by the application needs and depends on the $\langle O, C\rangle$ world plane. However, as no object in O has access to any clocks, the implementation design space depends on the $\langle P, L\rangle$ observation plane and how it interacts with the $\langle O, C\rangle$ plane.

3.1 Design Space for Specifying Timing Properties

1. Specification of time modality on predicate:
   (a) Single time axis (interleaved model):
      i. *Instantaneous*: This is the most popular and relevant specification in the literature for observing the world plane events that occurred at the same instant in physical time. Applications in pervasive systems aim to observe the instantaneous system state and draw inferences about it, e.g., raise alarm and context determination and context switch [4, 5, 16, 30, 36, 37].
      In the traditional distributed computing literature, Mayo and Kearns gave an algorithm to detect distributed predicates that held at some instant in time in a system using approximately synchronized physical clocks [28]. Stoller [34] likewise gives an algorithm to detect global state predicates with approximately-synchronized real-time

clocks. In both approaches, predicates specified using the *Instantaneously* modality on the execution events are detected using a physical time reference.

ii. Relative timing relations: Some attempts have been made at specifying such constraints for real-world observation [22, 29], using the theory developed for uniprocessor systems [1, 15]. Examples are: X before Y, or X overlaps Y, or X before Y by real-time greater than 5 seconds. An example from secure banking is [22]: a biometric key is presented remotely after a password is entered across the network.

iii. Physical time reference: Real-time applications may use such wall-clock specifications, e.g., after 7 o'clock.

iv. Temporal logic (*TL*) based: see [6] for a recent survey.

Combinations of the above can also be constructed.

(b) Multiple time axis (partial order model):

i. Causality based relations: In traditional distributed systems literature, there is a vast body of work on modalities based on causality-based relations. The *Possibly* and $Definitely$ modalities [10] have been the most widely used. Refining these further, a complete suite of 40 orthogonal relationships among time intervals at two different physical locations (see [7, 8, 20, 21]) was used to specify causality-based relationships among the local values that held during the local time intervals. Then, given a system with n processes, a specification space of size $(2^{40}-1)C_2^n$ for fine-grained relationships was identified.

In the context of sensornet applications, these modalities are currently being investigated. An application that requires simple concurrency detection is given in [17]; here, the $Definitely$ modality on a conjunctive predicate was detected using the partial order model. Consider a smart office environment where a person enters a room and $temp > 30°C$. Temperature can be automatically lowered depending on the rule base. However, no compelling applications that require such partial order specifications have yet been built.

ii. Physical time reference: A partial order model of time using a physical time reference for each location can also be used, e.g., to represent the physical time of the latest update to the versions of a file. However, physical time reference for a multiple time axis model is not popularly used in traditional distributed systems, and not so in sensor networks either.

iii. Temporal logic (*TL*) based: convincing application specifications to observe world plane executions using temporal logic are under study [6].

2. Predicate type: Though there exist many classes of predicates [9, 23], two interesting classes for observing world plane executions are listed here.

(a) Conjunctive: Each conjunct ϕ_i in the predicate $\phi = \bigwedge_i \phi_i$ can be locally evaluated by a process using local variables [14]. The subscript on a variable denotes the location where the variable is sensed. For example, $\psi =_{def} (x_i = 5) \wedge (y_j > 7)$, where x_i and y_j are the number of objects in rooms i and j, respectively; and $\chi =_{def} (temp_i = 20C \wedge person\_in\_room_i)$, are conjunctive.

(b) Relational: In contrast, a *relational* predicate ϕ is an arbitrary expression on the system-wide sensed variables [10]. For example, $\phi = x_i + y_j > 7$, where x_i and y_j are as defined above.

3.2 Design Space for Implementing Time

1. Clocks:

(a) The single time axis (interleaved model) using which predicates are specified (Section 3.1.1.a) can be implemented in the following ways.

   i. Perfectly synchronized physical scalar clocks. This is the ideal case, assumed by most of the pervasive computing community [4, 5, 16, 30, 36, 37] (except [17]), but not practical.
   ii. Imperfectly synchronized (with skew/offsets) physical scalar clocks: There is a vast literature in the last decade for implementing tightly synchronized clocks at low cost in wireless sensor networks; see [31], or a comprehensive survey in [35]. We note two important points here. (1) This service does not come for free to the application; the lower layers pay the cost. (2) The issues of drift/skew minimize but cannot eliminate the uncertainty in the face of race conditions when events happen very close in time in the physical plane.
   iii. Logical (asynchronous) scalar clocks: These are of the Lamport flavor [26] for traditional distributed systems, and defined formally for sensor networks in [25]; Though widely used in distributed systems, they have been used in [25] for observing the world plane events. In [25], it was shown how to use logical asynchronous scalar clocks to simulate the single time axis model for a physical time modality, *Instantaneously*.
   iv. Logical (asynchronous) vector clocks: These are of the Mattern flavor [27] in traditional distributed systems, and have been used in sensor networks [24] to track intervals at various processes to simulate the single time axis model. Further, in [25], it was shown how to use logical asynchronous vector clocks to simulate the single time axis model for a physical time modality, *Instantaneously*.

   We compare the methods described in Section 3.2.1.a(ii)-(iv) to implement the single time axis model below in Section 3.3.

(b) The multiple time axis (partial order model) using which predicates are specified (Section 3.1.1.b) can be implemented in the following ways:
   i. Logical (asynchronous) vector clocks: These are of the Mattern [27] and Fidge [13] flavor to track *causality* and capture the partial order of network plane events. The strobe vector clock given in [25] showed how to use vector time for sensing physical world events in the partial order time model. The strobe vector clock is limited to observing world plane events.
   Vector clocks were also used for concurrency detection of world plane events using the partial order model of time [17] to determine contextual properties.
   ii. Physical (asynchronous) vector clocks: These vectors use the monotonic physical (local) unsynchronized clocks of the processes as the vector components. These seem an overkill to track causality, but are useful when relating the locally observed wall times at different locations, in the application predicate.

2. Message (transmission and propagation) delay:
   (a) Instantaneous or synchronous: Ideal case.
   (b) Asynchronous Δ-bounded: The Δ-bounded transmission delays can often be assumed in practical asynchronous wireless networks because although there is variability in scheduling for energy conservation, the delay is bounded due to the bounded number of attempts at retransmissions. This model is practical in many cases in sensor-actuator networks and a good approximation to the ideal case when Δ is small relative to the rate of occurrence of events in the world plane [25, 24]. The Δ-bound has been used for quantifying errors here.
   (c) Asynchronous unbounded: Good for a worst-case analysis.

3.3 Options to Implement Single Time Axis Model

We now compare the trade-offs among the options in Section 3.2.1.a.(i)-(iv) to implement the single time axis model. Perfectly synchronized physical clocks, as assumed by the pervasive computing literature [4, 5, 16, 30, 36, 37], are impractical. Clearly, imperfectly synchronized physical clocks are desirable because clock synchronization is quite accurate. Skews of the order of nanosec are achievable. For sensor networks, there are many protocols tailored for the resource-constrained environments, e.g., RBS, TPSN, TinySync, TSync, that achieve skews of the order of microsecs to millisecs [35]. However, we note some limitations of this option.

1. Observe that no physically synchronized clock service may be available from a lower layer. This might be the case for very resource-constrained embedded sensors or those in remote environments in the wild. Furthermore, even if it is available, it may not be affordable (in terms of energy consumption), e.g., consider the wild or remote terrain. We stress that such service is not for free.

2. Physically synchronized clock service also has a skew ϵ and drift bounds, which leads to imprecision in detecting predicates in physical time. Predicate detection is prone to false negatives and false positives when there are "races" in sensing different physical world properties. A "race" occurs when two or more events occur at different locations and it is not possible for a global observer to determine the physical time ordering of the events. It has been shown that when the overlap period of the local intervals, during which the global predicate is true, is less than 2ϵ, false negatives occur [28].

3. Physically synchronized clock service imposes cross-layer dependence.

4. Even if physically synchronized clock service is available, the level of accuracy it provides may not be needed when dealing with slow human and object movement speeds. Software clocks can be used instead.

5. Physically synchronized clock service imposes security and privacy concerns by requiring all the participating entities from different domains to synchronize their clocks. A user may be unwilling to let his mobile device participate in clock synchronization.

In a scenario where the above limitations come into play, resulting in the absence of a synchronized physical time scale from a lower layer, how do we simulate a single time axis model? Here, to *re-create* a linear order time base in order to timestamp events, the application is forced to implement either

- an application-layer software to synchronize the physical clocks of sensors/ actuators, or

- synchronized logical clocks – either scalar (for single time axis model) or vector (for multiple time axis model).

The first option is not appealing because it is essentially incurring the overheads of the lower layer clock synchronization – which may not even be possible. Therefore, we advocate the latter option. Earlier in [25], we explored the option of using lightweight middleware protocols using *strobe clocks* [25], without accessing physically synchronized clock service, to detect global predicates. We showed that the accuracy of detecting predicates is affected, resulting in false negatives, and possibly false positives, when races occur within a period of Δ [24, 25]. Logical vector clocks provide more accuracy than logical scalar clocks. In particular, the use of logical vectors may result in some false negatives, whereas the use of logical scalars may also result in some false positives. Strobe clocks are reviewed and discussed further in Section 4.2.

Δ, that determines the accuracy of the algorithms [24, 25], is of the order of hundreds of millisecs to secs in small-scale networks, such as smart offices and smart homes. Even though Δ is much larger than the ϵ skew (microsecs to millisecs) that determines the accuracy of predicate detection using synchronized physical clocks, Δ may be adequate when (a) the number of processes is low and/or (b) the rate of occurrence of sensed events is comparatively low. The latter is the case for several environments in the urban setting (such as office, home, and structure monitoring) and in the wild (such as in habitat, wildlife, nature monitoring). Lifeform and physical object movements are typically much slower than Δ. In the above identified urban settings, and in the wild, remote terrain, nature monitoring, events are often rare, compared to Δ. Thus, we may not need the precision of

synchronized physical clocks (both, in urban settings or in the wild) nor may we be able to afford the associated cost of synchronized physical clocks (in the wild). Simulations in related work [17] to detect $Definitely(\phi)$ for a conjunctive ϕ in a realistic model of a smart office showed that despite increasing the average message delay over a wide range, the probability of correct detection is quite high. The simulations were backed by an analytical model with supporting numerical results.
Example: As an example of formalizing a specific problem using the design space for specification of timing properties, and the design space for implementing time, we consider the problem of detecting a relational predicate of observed world properties, that held at some instant in physical time.
Problem Specification: Detect *each* occurrence of a predicate ϕ on sensed attribute values of the world plane, where:

- Time modality of predicate: is Single time axis – Instantaneous
- Predicate type: is Relational
- Message delay: is asynchronous Δ-bounded
- Clocks: may be either single time axis (interleaved model) with logical (asynchronous) scalar clocks, or multiple time axis (partial order model) with logical or physical (asynchronous) vector clocks.

This problem specification was addressed and solved in [25]. We emphasize that each occurrence of the predicate should be detected. For example, (i) reset thermostat to $28 \deg C$ each time "motion detected" $\wedge$ "temp $> 30 \deg C$"; and (ii) lock office door each time "no motion detected" $\wedge$ "lights off". Existing literature on predicate detection, e.g., [14, 17], detects only the first time the predicate becomes true and then the algorithms "hang".

In Section 4, we examine the suitability of the single time axis model versus the multiple time axis model for pervasive environments with sensor-actuator networks.

4 Time Models for Pervasive Systems

The single axis time model is useful for specifying timing properties in sensor-actuator pervasive networks, as shown in Section 3.1.1.a. In Section 4.1, we examine the validity of the partial order model of time in specifying timing properties in sensor-actuator pervasive networks.

4.1 Partial Order Model as a Specification Tool

The partial order of time, captured by vector clocks, is necessary, even if synchronized physical clocks are present, to capture the cause-effect dependency relationships among events. This is the first use of modeling the partial order of time and events. The partial order of the traditional distributed program execution in the network plane induced by the causality relation is isomorphic to and captured by the causality-based Mattern/Fidge clocks [13, 27]. The notion of "alternate global states that could have occurred in an equivalent execution (due to the asynchrony in process execution and message communication)" leads to the notion of the lattice of possible global states and its sublattice of consistent global states [10, 27]. Reasoning about repeated runs of deterministic distributed executions in terms of the state lattice is the second use of modeling the partial order of time and events.

In state-of-the-art pervasive systems, there are major differences from in-network distributed program executions. Consider world events $a@t1@l_i$ and $b@t2@l_j$ (using $event\_label@global\_time@location$ format). Is there a causal dependence from a to b (a la Lamport [26])? If we could track the "hidden channel" communication between the events and the semantics of this communication occurring in the $\langle O, C\rangle$ plane, we can answer this and simulate it exactly in the $\langle P, L\rangle$ network plane to maintain our evolving "map" of the world plane.

Consider a smart office, where each object has embedded intelligence. When Bob gives a pen to Tom, Tom then moves to another room, and leaves the pen there, the physical handoff and transport of the pen can be detected by all the sensors/ badge readers. The causality from event $pen@t1@l_i \longrightarrow$ event $pen@t2@l_j$ in the world plane can be tracked in the network plane. We can mirror the physical world causal chain in our virtual map of it. But one could argue, if the pen is intelligent and not just embedded with a RFID tag, it is part of the network plane also, not just of the world plane. If the pen were dumb, sophisticated motion detectors and pattern recognition software could rebuild the entire causal chain in the physical world at great cost, but this does not seem practical in a more general setting. Thus, presently, technology does not allow tracking of the hidden channels and causality chains in the general case.

Some other examples are: (1) wind spreading a forest fire, (2) Bob posting a letter in the red postbox on the road, Tom in another city receiving the letter two days later and acting on it. This limitation has been expressed earlier, see [18, 32, 33]. We cannot always determine concurrency among world plane events because we cannot always monitor (due to current limitations) the communication in the covert channels. Thus, if we could track this causality exactly, it would make sense to use the partial order model in the specification of the physical world map. If the partial order is defined by the causal relation (as defined by Lamport [26]), it can be used (i.e., implemented) if an application predicate is specified using it. The authors are presently unaware of deployed applications that use the partial order to track true causality/concurrency in the world plane. So on this count, there is no case yet for using the partial order model of time as a specification tool for predicates in sensornets.

Note that in distributed programs, there is a second use of the partial order of time – to create the partial order lattice of the states of the execution. As noted above, the global state lattice constructed based on the causality-based partial order of time, is useful to reason about properties of global states. This reasoning is across all runs of the same deterministic distributed program; not just for one run. In a re-run, concurrent events may be reordered (due to the asynchrony in message transmission times and process scheduling), leading to a different path in the state lattice. However, in a pervasive environment, the physical world does not admit re-runs, and there are many *non-deterministic factors* such as human will and nature. Further, usually most applications need to observe the actual states in the actual execution as time unfolded. Therefore, the state lattice seems not needed; in fact, the state lattice is the lattice of p^n states that admits all possible concurrent states, because the network plane cannot capture the dependencies of the world plane. Thus, the state lattice becomes effectively meaningless unless the network plane can capture the true causal dependencies of world events, that set in through the hidden channels in the world plane, and that need to be simulated in the network plane. Therefore, there is no case yet for using the partial order model of time as a specification tool for a distributed predicate.

The only communication through the network plane that effects causality in the world is a sequence like: $e1@l1 \longrightarrow sense@l1 \longrightarrow actuate@l2 \longrightarrow e2@l2$. We still need to answer whether there was true causality between $e1$ and $e2$ in the same sense that there is causality between statements $x := 5$ and $z := f(y)$ in this distributed program:
$P_1 : \ldots x := 5; send(x, P_2); \ldots$
$P_2 : \ldots receive(y, P_1); z := f(y); \ldots$
The moot point is: Can the lower network plane $\langle P, L\rangle$ be interlocked or meshed in with the upper plane $\langle O, C\rangle$, not just have a "bridge" to it? That is to say, how successfully can the lower plane not just observe but also actuate and control the upper plane? A robotic network in a warehouse (a confined setting) is an example of such a real system that attempts to intermesh with the world plane.

The partial order time model thus has not been seen to be useful as a specification tool in real applications of sensornets. Yet, as we show now in Section 4.2, to simulate the linear order time model, the partial order time model is useful for real applications in pervasive sensornets.

4.2 Partial Order Model as a Implementation Tool

Logical time need not be based strictly on causality as defined by message-passing at the application layer. A need for building a partial order of time that is useful for observing the world plane events under the *Instantaneously* modality of physical time was shown in [25]. In this section, we review this need and the proposed strobe clocks [25] to address this need. In the absence of physically synchronized clock service, some time base is needed. The idea is that logical time can simply be used to provide a base of linear order/ partial order time when physical clocks are not required (due to cost or layer independence or over-accuracy) or not available. Observe that lower network layer physical clock synchronization protocols [35] periodically bring multiple hardware clocks (scalars) "in sync" after some drift. Similarly, the application layer *strobe clock* can periodically bring "in sync" the drifting scalars or vectors at each process. In the absence of a strobe, logical clocks drift, simply ticking asynchronously at each relevant local event. Each process maintains a local clock component that ticks asynchronously. The strobe clock is a logical (scalar or vector) clock synchronization service to synchronize the local clocks at "critical events". The strobe clock needs to guarantee monotonicity of logical time. The strobe by a process can synchronize at any time. However, this synchronization need not happen any more frequently than the local sensing of relevant events.

A typical protocol for physical clock synchronization that operates in this manner is [3]. The protocol performs on-demand clock synchronization and messages required for continuous synchronization are avoided. Nodes do not share the same time basis. For example, nodes only need to start a common task at a certain point in time, but do not need a common time basis apart from that. The network stays unsynchronized most of the time but collaborates shortly before the common event. An application is the collaborative sensing of highly dynamic effects, e.g., locating the source of an audio signal, or simultaneous playback of music by the sensor/actuator network.

Vector clocks were used in [17] to implement predicate detection in the partial order model of time in pervasive sensornets to determine context.

Strobe clock messages [25] are control messages and induce a partial order that is arbitrarily determined at run-time and hence artificial. This is in contrast to the case for distributed programs, where the partial order is induced explicitly by in-network semantic send and receive events of the programs. Note, if our map of the physical world is also tracking causality, that clock should necessarily be different from the strobe clock. If it is not, it will introduce false causality induced by the strobes. This will lead to inferring fake causal dependency relationships among computation and actuator events (e and a events) in our model given in Section 2.2. This will also eliminate possible equivalent consistent global states.

4.2.1 Strobe Vector Clocks

A strobe vector clock $C_i[1..n]$ at process i consists of n integers. The *protocol* is given by rules SVC1 and SVC2.

```
SVC1. When process $i$ executes (senses) a relevant event:
    $C_i[i] = C_i[i] + 1$
    System-wide_Broadcast ($C_i$)

SVC2. When process $i$ receives a strobe $T$:
    $(k \in N)\ C_i[k] = \max(C_i[k], T[k])$
```

In contrast, a causality-based Mattern/Fidge vector clock $C_i[1..n]$ at process i consists of n integers. The *protocol* is given by rules VC1 – VC3 [13, 27].

```
VC1. When process $i$ executes (senses) a relevant internal event:
    $C_i[i] = C_i[i] + 1$

VC2. When process $i$ executes a send event to send message $M$:
    $C_i[i] = C_i[i] + 1$
    Send M($C_i$)
```

VC3. When process i receives a vector T piggybacked on a message:
$(k \in N)$ $C_i[k] = \max(C_i[k], T[k])$
$C_i[i] = C_i[i] + 1$

Note that the Mattern/Fidge vector clock protocol has no occasion to send an execution message M, and invoke rules VC2 or VC3 when observing world plane events.

4.2.2 Strobe Scalar Clocks

A strobe scalar clock C_i is maintained by each process i. The *protocol* is given by rules SSC1 and SSC2.

SSC1. When process i executes (senses) a relevant event:
$C_i = C_i + 1$
System-wide_Broadcast (C)

SSC2. When process i receives a strobe T:
$C_i = \max(C_i, T)$

It is weaker than the strobe vector clock but is lightweight (strobe size is $O(1)$, not $O(n)$).

In contrast, a logical scalar Lamport clock C_i is maintained by each process i. The *protocol* is given by rules SC1 – SC3 [26].

SC1. When process i executes (senses) a relevant event:
$C_i = C_i + 1$

SC2. When process i executes a send event to send message M:
$C_i = C_i + 1$
Send M(C_i)

SC3. When process i receives a scalar timestamp T piggybacked on a message:
$C_i = \max(C_i, T)$
$C_i = C_i + 1$

Again, note that the Lamport clock protocol has no occasion to send an execution message M, and invoke rules SC2 or SC3 when observing world plane events.

Strobe clock protocols, whether scalar or vector, use broadcasts. A message loss may result in the wrong detection of the predicate in the temporal vicinity of the lost message. However, there will be no long-term ripple effects of the message loss on later detection.

4.2.3 Comparing Strobe Clocks with Causality-based Clocks

The logical strobe clocks (vector and scalar) differ from the traditional causality-based Mattern/Fidge vector clocks and Lamport scalar clocks in the following ways.

1. Strobe clocks track the progress of the local logical time counter at each process by catching up or synchronizing on the latest known time of other processes. They do not track the causality induced by message communication. Causality-based clocks track the causality induced by the $\langle N, L\rangle$ -plane in-network message sends and receives.

2. On receiving a strobe, the receiver updates its clock but does not tick locally; in causality-based clocks, the receiver ticks on receiving a message.

3. All strobes are control messages. In causality-based clocks, timestamps are piggybacked only on all computation messages.

4. The strobe clock protocol broadcasts its clock no more frequently than at each relevant event (after ticking its local component). In causality-based clocks, the clock value is piggybacked only on all computation messages. For vector clocks, this creates an isomorphism of the partial order of events.

5. When synchronous communication is used, i.e., when $\Delta = 0$, and the protocol strobes at each relevant event, strobe vectors can be replaced by strobe scalars without sacrificing correctness or accuracy. This is not so for the causality-based clocks even if $\Delta = 0$; Mattern/Fidge clocks are still more powerful than Lamport clocks when reasoning about the partial order of distributed program executions.

4.2.4 Simulating Linear Time Model

The physical world $\langle O, C\rangle$ plane execution traces one path through np of the $O(p^n)$ states in the state lattice. Ideally, the states in this path should be identified so that the predicate can be evaluated in each of them. Although the control messages for the strobe clock create artificial causal dependencies, these are useful because they help to approximate instantaneous observation by eliminating many of the $O(p^n)$ states in which the corresponding intervals did not overlap. However, the number of possible consistent states in the sub-lattice induced by the strobes is still $O(p^n)$. The faster the strobe transmissions, the leaner is the lattice. When $\Delta = 0$, the result is a linear order of np states. Observe that in executions of distributed programs, program-determined semantic messages may not get sent for long durations, resulting in fat lattices. In contrast, clock strobes get sent each time a sensed value changes. This gives the "slim lattice postulate" [25] for consistent global states in sensornet observations. Algorithms using vector strobes and scalar strobes to detect global predicates based on sensed world properties are given in [24] and [25].

5 Application Scenarios

Consider a convention center where entry tickets serve as RFID badges for the visitors. Consider a big exhibition hall within the convention center. The exhibition hall has d doors for entry-cum-exit, and has a room capacity of 200 people. At each door, a sensor detects the movement of people in and out of the hall by using RFID scanning of the attendees' convention badges. Each sensor is modeled as a process P_i and tracks two variables: x_i, the number of people entered through the monitored door, and y_i, the number of people that have left through the monitored door. A sensing event at a sensor process P_i is the entry or exit of a person through the door that the sensor monitors. The global predicate that we seek to monitor under the *Instantaneously* modality is $\phi = \sum_{i=1}^{d}(x_i - y_i) > 200$ over the data sensed by all the d sensors. When the predicate becomes true, entry into the hall is not allowed, until the predicate becomes false again. Detection of this predicate is required to prevent overcrowding and violation of fire code norms.

Although physically synchronized clocks could be implemented in this urban setting, their overhead is not necessary because the precision they provide is more than required for detecting human movements in this detection problem. Using the proposed logical strobe clocks, the algorithms given in [24] can detect the predicate ϕ. For the network plane $\langle P, L\rangle$, the processes in P are the sensors and the logical wireless links connecting them form L. Essentially, the sensor processes run the vector strobe clock algorithm among themselves to recreate a linear time base. Thus, the partial order time model is used to simulate the linear order time model. A broadcast is done on the occurrence of each sensed event. Due to a race condition when there is concurrent traffic through multiple doors, and due to variations in transmission delay, a false negative may occur when the occupancy is above 200, and a false positive may occur when the occupancy is below 201. This is within acceptable limits of tolerance. Furthermore, the consensus based algorithm using vector strobes will be able to place false positives and most false negatives in a "borderline bin" which is characterized by a race condition. The application can treat entries in the borderline bin as positives or negatives. To err on the safe side, such entries can be treated as positives.

As another example, consider a hospital where each visitor and patient has a RFID badge. Analogous to the above exhibition hall scenario, we could monitor the number of visitors in the waiting room. Or when a visitor enters the infectious diseases ward. Or we could raise alarms when a visitor approaches a patient whom he is not visiting. There are numerous such scenarios that we can model, specify predicates on such scenarios, and detect those predicates using vector strobe clocks.

At the lower nework layer level, synchronization of duty cycles among wireless sensor nodes for efficient execution of MAC and routing layer functions can be achieved using distributed timers. It is particularly feasible in applications such as habitat monitoring where the monitoring activities proceed slowly. Using the proposed execution model, synchronization can be achieved via send and receive events.

6 Discussion

We proposed a general system model and an execution model for sensor-actuator networks in distributed pervasive environments. We charted out a time model space for specifying properties to be detected, and a time model space for implementing the specifications. Next, we examined the range of time models that are useful for such distributed sensor networks, and placed approaches and limitations in perspective.

We conclude that presently, the partial order time model for specification of predicates has not found adequate uses in pervasive sensor-actuator networks. Rather, the single axis time model continues to be widely used. On the other hand, we also explored the options of implementing the single axis time model. While physical clock synchronization protocols are clearly a desirable option to provide the single time axis, we showed that in some applications (characterized by the unavailability and or the high cost of such clock synchronization protocols), logical time strobe vector clocks that provide a partial order of time are a viable alternative, particularly when: (i) the s (sensing) event occurrence rate is low with respect to Δ, or (ii) physical synchronized clocks are too expensive or not available or needed. The ultimate test for this depends on their incorporation in useful pervasive sensornet applications.

We identify two directions for further investigation.

- The use of the partial order model of time as a specification tool seems to be limited due to the inability to track causality in the world plane due to the hidden channels. However, there are likely to be some applications where the world plane communications can be tracked by the network plane. Such applications should be investigated. The partial order time model will be a natural fit for such distributed applications, e.g., a secure banking application where the use of concurrent biometric passwords from remote locations is used for authentication [22].
- The use of the linear order model of time as a specification tool can be addressed naturally by using the linear order of time as an implementation tool, viz., using physically synchronized clocks. However, the conditions described in Section 3.3 seem to make the strobe clocks, i.e., partial order of time as an implementation tool, a viable alternative to simulate the linear order of time. A study of real sensornet applications is required to evaluate the viability.

Acknowledgement

This work was supported by National Science Foundation grant CNS 0910988, “Context-Driven Management of Heterogeneous Sensor Networks.”

References

[1] J. Allen, Maintaining knowledge about temporal intervals, *Communications of the ACM*, 26(11): 832-843, 1983.

[2] O. Babaoglu, K. Marzullo, Consistent global states of distributed systems: fundamental concepts and Mechanisms, In: S. Mullender (ed.) *Distributed Systems*, Chapter 5, 97-145, Addison-Wesley, 2nd edition, 1993.

[3] T. Baumgartner, S. P. Fekete, W. Hellmann, A. Kroller, Simultaneous event execution in heterogeneous wireless networks, *Journal of Networks*, 5(10): 1221-1226, 2010.

[4] Y. Bu, T. Gu, X. Tao, J. Li, S. Chen, J. Lu, Managing quality of context in pervasive computing, In *Proc. International Conference on Quality Software*, 193-200, 2006.

[5] Y. Bu, S. Chen, J. Li, X. Tao, J. Lu, Context consistency management using ontology based model, In *Proc. Current Trends in Database Technology*, 741-755, 2006.

[6] R. Cardell-Oliver, M. Renolds, M. Kranz, A space and time requirements logic for sensor networks, In *Proc. Second International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation*, 283-289, 2006.

[7] P. Chandra, A. D. Kshemkalyani, Causality-based predicate detection across space and time, *IEEE Transactions on Computers*, 54(11): 1438-1453, 2005.

[8] P. Chandra, A.D. Kshemkalyani, Data stream based global event monitoring using pairwise interactions, *Journal of Parallel and Distributed Computing*, 68(6): 729-751, 2008.

[9] C. M. Chase, V. K. Garg: Detection of global predicates: techniques and their limitations, *Distributed Computing*, 11(4): 191-201, 1998.

[10] R. Cooper, K. Marzullo, Consistent detection of global predicates, In *Proc. ACM/ONR Workshop on Parallel and Distributed Debugging*, 163-173, May 1991.

[11] J. Elson, D. Estrin, Sensor networks: a bridge to the physical world, In: *Proc. Wireless Sensor Networks*, Kluwer Academic Publishers, 2004.

[12] D. Estrin et al., Instrumenting the world with wireless sensor networks, In *Proc. ICASSP*, 2001.

[13] C. Fidge, Timestamps in message-passing systems that preserve partial ordering, *Australian Computer Science Communications*, 10(1): 56-66, Feb. 1988.

[14] V. K. Garg, B. Waldecker, Detection of strong unstable predicates in distributed programs, *IEEE Trans. Parallel and Distributed Systems*, 7(12):1323-1333, Dec. 1996.

[15] C. L. Hamblin, Instants and intervals, in *"The Study of Time,"* pp. 324-332, Springer-Verlag New York/Berlin, 1972.

[16] P. Hu, J. Indulska, R. Robinson, An autonomic context management system for pervasive computing, In *Proc. IEEE International Conference on Pervasive Computing and Communications (Percom)*, 213-223, 2008.

[17] Y. Huang, X. Ma, J. Cao, X. Tao, J. Lu, Concurrent event detection for asynchronous consistency checking of pervasive context, In *Proc. IEEE International Conference on Pervasive Computing and Communications*, 2009.

[18] L. Kaveti, S. Pulluri, G. Singh, Event ordering in pervasive sensor networks, In *Proc. IEEE International Conference on Pervasive Computing and Communications Workshops*, 2009.

[19] A. Khelil, F. Shaikh, B. Ayari, N. Suri, MWM: a map-based world model for wireless sensor networks, In *Proc. Autonomics*, 2008.

[20] A. D. Kshemkalyani, Temporal interactions of intervals in distributed systems, *Journal of Computer and System Sciences*, 52(2): 287-298, April 1996.

[21] A. D. Kshemkalyani, A fine-grained modality classification for global predicates, *IEEE Trans. Parallel and Distributed Systems*, 14(8): 807-816, August 2003.

[22] A.D. Kshemkalyani, Temporal predicate detection using synchronized clocks, *IEEE Transactions on Computers*, 56(11): 1578-1584, November 2007.

[23] A.D. Kshemkalyani, M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*, Cambridge University Press, 2008.

[24] A.D. Kshemkalyani, Immdiate detection of predicates in pervasive environments, In *Proc. 9th International Workshop on Adaptive and Reflective Middleware*, 18-25, ACM Press, 2010.

[25] A.D. Kshemkalyani, Middleware clocks for sensing the physical world, In *Proc. 5th International Workshop on Middleware Tools, Services, and Run-Time Support for Sensor Networks*, 15-21, ACM Press, 2010.

[26] L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM*, 21(7): 558-565, July 1978.

[27] F. Mattern, Virtual time and global states of distributed systems, In *Parallel and Distributed Algorithms*, North-Holland, pp 215-226, 1989.

[28] J. Mayo, P. Kearns, Global predicates in rough real time, In *Proc. IEEE Symp. on Parallel and Distributed Processing*, 17-24, 1995.

[29] P. Pietzuch, B. Shand, J. Bacon, Composite event detection as a generic middleware extension, *IEEE Network*, 18(1): 44-55, Jan/Feb 2004.

[30] M. Roman, C. Hess, R. Cerqeira, A. Ranganathan, R.H. Campbell, K. Nahrstedt, A middleware infrastructure for active spaces, *IEEE Pervasive Computing*, 1(4): 74-83, 2002.

[31] K. Romer, Time synchronization in ad-hoc networks, In *Proc. ACM MobiHoc*, 2001.

[32] K. Romer, F. Mattern, Towards a unified view on space and time in sensor networks, *Computer Communications*, 28(13):1484-1497, August 2005.

[33] K. Romer, F. Mattern, Event-based systems for detecting real-world states with sensor networks: A critical analysis, In *Proc. DEST Workshop on Signal Processing in Wireless Sensor Networks at ISSNIP*, pp. 389-395, Melbourne, Australia, December 2004.

[34] S. Stoller, Detecting global predicates in distributed systems with clocks, *Distributed Computing*, 13:85-98, 2000.

[35] B. Sundararaman, U. Buy, A. D. Kshemkalyani, Clock synchronization for wireless sensor networks: A survey, *Ad-Hoc Networks*, 3(3): 281-323, May 2005.

[36] C. Xu, S.C. Cheung, Inconsistency detection and resolution for context-aware middleware support, In *Proc. ACM SIGSOFT Int. Symposium on Foundations of Software Engineering*, 336-345, 2005.

[37] C. Xu, S.C. Cheung, W.K. Chan, Incremental consistency checking for pervasive context, In *Proc. International Conference on Software Engineering*, 292-301, 2006.

Appendix A

Traditional asynchronous message-passing distributed systems use two common time models:

1. The single time axis model, that loosely corresponds to the event interleaving model of the distributed execution. The single time axis model can be implemented by either tightly synchronized physical clocks or by Lamport's logical clock.

   (a) At the network protocol layer and the operating system layer, tightly synchronized physical clocks are used for various uses such as: reference to physical time (Universal Coordinated Time: UTC) for tracking activities, and for synchronization protocols at the MAC layer.

   (b) At the higher layers including the middleware and the application layer, physical clocks are used for reference to the physical time of occurrence of events and of composite events at a process that effectively demarcate a time interval.

   (c) There are no popular uses of the logical clock at the network protocol layer and the operating systems layer.

   (d) Lamport's logical clock is used mainly at the middleware and the application layer to relatively order events to determine causality between them; for example, to enforce mutual exclusion across the distributed system or to satisfy fairness of requests for access to resources. This concept can be extended to define composite events at a process that effectively demarcate a time interval.

2. The vector time model, that loosely corresponds to the partial order model of events in the distributed execution. The vector time model can also be implemented by either tightly synchronized physical clocks or by logical clocks (Mattern/Fidge's vector clocks).

   (a) There are no popular uses of vector time (using physical clocks) at the network and operating systems layer.

   (b) At the higher layers including the middleware and application layer, vector clocks based on physical clocks can track the exact physical time of the occurrence of events at other processes, such that those events were the latest events at those processes to causally affect the current state at the process under consideration. But what the vector clock really does is capture the partial order of the execution by tracking the causality relationship using physical time instead of relative time. The reliance on the use of physical clocks to track causality is an overkill (unless the physical time of causally preceding remote events is also to be tracked) and runs the risk of erroneous conclusions due to skew and drift among the physical clocks.

   (c) There are no popular uses of vector time using the logical vector clock at the network protocol layer and the operating systems layer.

   (d) The vector clock based on logical time has numerous applications at the middleware and application layers, wherever it is useful to track the partial order of the distributed execution. Traditional applications include: checkpointing, garbage collection, causal memory, maintaining consistency of replicated files, taking efficient consistent snapshots of a system, global time approximation, termination detection, bounded multiwriter construction of shared variables, mutual exclusion, debugging, and defining concurrency measures. These are well documented in the literature. Emerging and recent areas that use vector clocks include building reliable massive-scale ecommerce systems, building software transactional memory, studying information spread in social communication networks, maintaining data consistency in collaborative peer-to-peer editing, dynamic race detection in multithreaded programs, and designing massive multiplayer online games.