Predicate Detection Using Event Streams in Ubiquitous Environments

Ajay D. Kshemkalyani

Computer Science Department, Univ. of Illinois at Chicago, Chicago, IL 60607, USA
ajayk@cs.uic.edu

Abstract. Advances in clock synchronization techniques for sensor networks as well as wireless ad-hoc networks allow an approximated global time for an increasing number of configurations in ubiquitous and pervasive computing environments. This paper presents an event stream based on-line algorithm that fuses the data reported from the heterogenous processors in the network to detect predicates of interest. The algorithm detects properties that can be specified using predicates under a rich palette of time modalities. The algorithm has low space, time, and message complexities. The main idea used to design the algorithm is that the predicate is decomposed as a collection of predicates between pairs of system devices. The algorithm leverages the *pairwise interaction* between processes so as to incur a low overhead and hence be highly scalable.

1 Introduction

Event-based data streams represent relevant state changes that occur at the processes that are monitored. The paradigm of analyzing event streams to mine data of interest to various applications uses *data fusion*. This paper gives an on-line algorithm to detect predicates from event streams that are reported by the various components of an ubiquitous computing environment. Such an environment includes ad-hoc networks and sensor networks [1, 19].

In the system model, the devices of the ubiquitous network are modeled by processes. The model assumes a loosely-coupled ad-hoc asynchronous message-passing system in which any two processes belonging to the process set $N = \{P_1, P_2, \ldots, P_n\}$ can communicate over logical channels. For a wireless communication system, a physical channel exists from P_i to P_j if and only if P_j is within P_i's range; a logical channel is a sequence of physical channels representing a multi-hop path. The only requirement is that each process be able to send its gathered data eventually and asynchronously (via any routes) in a FIFO stream to a data fusion server P_0.

E_i is the linearly ordered set of events executed by process P_i in an execution. Variable x local to process P_i is denoted as x_i. Given a network-wide predicate on the variables, the *intervals* of interest at each process are the durations during which the local predicate is true. Such an interval at process P_i is identified by the (totally ordered) corresponding adjacent events within E_i, for which the local predicate is true. Intervals are denoted by capitals X, Y, and Z. The types

T. Enokido et al. (Eds.): EUC Workshops 2005, LNCS 3823, pp. 807–816, 2005.

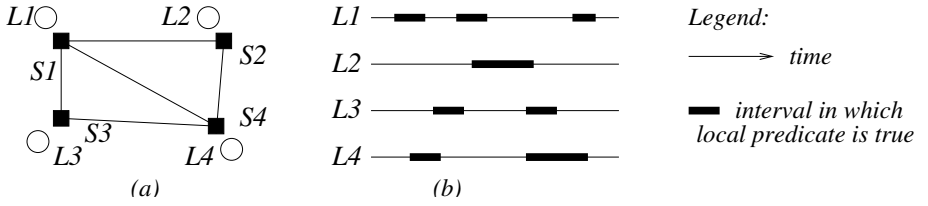

Fig. 1. Intervals within an ubiquitous network. (a) A network. $S1$ – $S4$ are sensors at locations $L1$ – $L4$. (b) Timing diagram for intervals at $L1$ – $L4$.

of predicates our algorithm handles are *conjunctive* predicates. A *conjunctive* predicate is of the form $\bigwedge_i \phi_i$, where ϕ_i is any predicate defined on variables local to process P_i. An example is: $(x_i > 4) \wedge (y_j = 94)$, where x_i and y_j are variables at P_i and P_j, respectively. Figure 1 shows four locations and intervals in their timing diagrams, during which the local predicates are true.

The problem we address is informally described as follows. Event streams from the processes report intervals in which the local predicates are true. Information about the reported intervals is "fused" or correlated and examined to *detect* global states of the execution that satisfy a given input predicate. This problem was defined and addressed earlier [3, 4, 5, 12] to detect predicates in distributed executions, using causality relationships defined in [11, 12].

Physical clocks in sensor networks, ad-hoc networks, and wireless networks – when synchronized via GPS [15], NTP [16], or any of the many efficient synchronization protocols for wired as well as wireless media, such as those surveyed in [8, 9, 17, 18] – allow the assumption about an approximate single global time axis. We assume such synchronized physical clocks. This assumption simplifies the detection of a global state [7] in the ubiquitous environment, which is essentially a form of a distributed asynchronous message-passing system. With synchronized clocks, a *distributed execution* is the interleaving of all the local executions E_i on a common time axis. A *global state* contains one local state of each process. Using a common time axis, a global state can be specified (i) as occurring at the same time instant at each process, or (ii) in terms of specific relationships among the local states (one local state from each processes).

For a single time axis, it has been shown [10, 2] that there are 13 ways in which two time intervals can be related to one another on that time axis. For intervals X and Y, the thirteen relations are:

- *precedes* and *preceded by* (which is $precedes^{-1}$)
- *meets* and *met by* (which is $meets^{-1}$)
- *overlaps* and *overlapped by* (which is $overlaps^{-1}$)
- *contains* and *contained by* (which is $contains^{-1}$)
- *starts* and *started by* (which is $starts^{-1}$)
- *finishes* and *finished by* (which is $finishes^{-1}$)
- *equals*

The set of these 13 relations is denoted $\Re$ and is illustrated in Figure 2. There are six pairs of inverses, and *equals* is its own inverse.

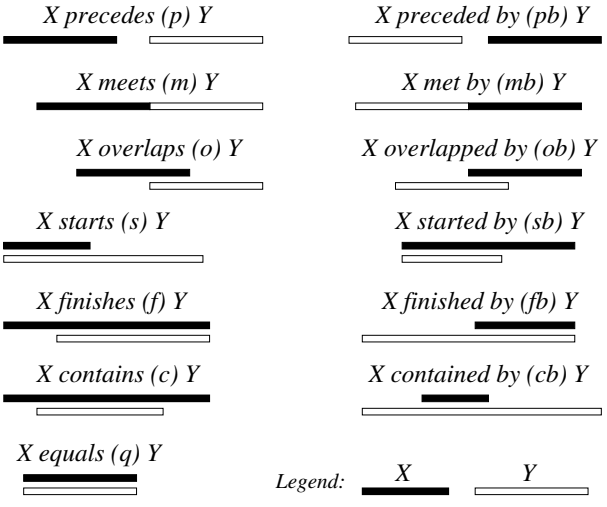

Fig. 2. The 13 relations $\Re$ between intervals

Our problem is now formally defined. Event streams generated by the different processors need to be fused at a central server to solve the following global predicate detection problem [3, 4, 5, 6, 12].

Problem *Predicate_Rel* **statement.** Given a relation $r_{i,j}$ from $\Re$ for each pair of processes P_i and P_j, identify the intervals (if they exist), one from each process, such that each relation $r_{i,j}$ is satisfied for the (P_i, P_j) pair.

Example specification: We assume that intervals X_i, Y_j, and Z_k occur at different locations i, j, and k, respectively, but global time is available in the system at all sites. Two example specifications of predicates are:

(a) (X_i precedes Y_j) AND (X_i overlaps Z_k) AND (Z_k finishes Y_j)
(b) (X_i overlaps Y_j) AND (Y_j contains Z_k) AND (Z_k met by X_i)

The problem in each case is to identify the global state in a distributed execution when the predicate is true. Example solutions are illustrated in Figure 3.

The performance of the proposed algorithm is summarized in Table 1. P_0 is the data fusion server that processes the event streams. The metrics are the time complexity at P_0, the network-wide count of the messages sent by the processes to P_0, the total space complexity at P_0, the cumulative size of all the messages sent to P_0, and the space at each process P_i. n = number of processes, p = maximum number of intervals occurring at any process.

2 Outline of the Algorithm

An interval at P_i begins when the local predicate ϕ_i becomes true and ends when ϕ_i becomes false. We assume the physical clock has infinitely fine granularity so

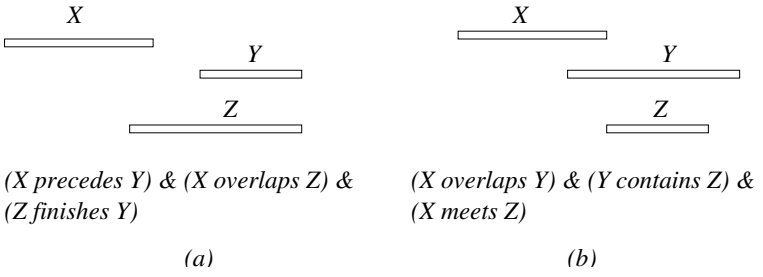

Fig. 3. Example problem specifications. The intervals X_i, Y_j, and Z_k are at different processes in the distributed ubiquitous system.

Table 1. Space, message and time complexities of the proposed algorithm

Time complexity at P_0	Total number of messages	Space at P_0 (=total message space)	Space at P_i, $i \in [1, n]$
$O((n-1)np)$	np	$2np$	2

each (event-triggered) state transition at a process occurs at a distinct tick (*local discreteness*). There are two consequences of *local discreteness* and the model for intervals. (1) An interval has a non-zero duration, implying *points* are not allowed. (2) An interval can begin at P_i only after the previous interval at P_i ends (see Fig. 1(b)) – termed the *local interval separation* property.

Processes P_1, P_2, ..., P_n representing the n devices in the ubiquitous network track the start and end timestamps of their local intervals. The timestamps are as per the synchronized physical clock based on a global time axis. t_i^- and t_i^+ denote the timestamps at process P_i at the start and at the end of an interval, respectively. This information is sent using the network asynchronously to the central data fusion server P_0. The only requirement is that between any process and P_0, the logical link must be FIFO. Recall that in a ubiquitous network, the numerous small devices operating collectively, rather than as stand-alone devices, form a dynamic ambient network that connects each device to more powerful networks and processing resources. Thus, the design using P_0 as the more powerful "server" fits this model.

The data fusion server maintains queues Q_1, Q_2, ... ,Q_n for interval information from each of the processes. The server runs the proposed algorithm to process the interval information it receives in the queues. For any pair of intervals, observe from Figure 2 that there is a overhead of $O(1)$ time complexity to test for each of the 13 relations using the start and end timestamps of the intervals. The algorithm detects "concurrent" pairwise interactions for each pair of intervals, considering only one interval from each process at a time as being a part of a potential solution. A challenge to solve *Predicate_Rel* is to formulate the necessary and sufficient conditions to determine when to eliminate the received information about intervals from the queues, so as to process the queues

type Log = **record**	Start of an interval:	End of interval:
$start$: integer;	$Log_i.start = t_i^-$.	$Log_i.end = t_i^+$
end: integer;		Send Log_i to central process P_0.
end		

Fig. 4. Data structures and operations to construct Log at P_i $(1 \leq i \leq n)$

efficiently. These conditions are important because intervals that are determined as not being part of a solution should not be used in testing with other intervals.

We assume that interval X occurs at P_i and interval Y occurs at P_j. For any two intervals X and X' that occur at the same process, if $precedes(X, X')$, then we say that X is a *predecessor* of X' and X' is a *successor* of X. The algorithm to solve problem *Predicate_Rel* is given in two parts. The processing on each of the n processes P_1 to P_n is given next. The processing by P_0 is given in Section 3.

Processing at P_i, $(1 \leq i \leq n)$

Each process P_i, where $1 \leq i \leq n$, maintains the data structure Log_i that contains the information of the start and end of the (latest) interval to be sent to P_0. Log_i is constructed and sent to P_0 using the protocol shown in Figure 4. P_0 uses the Logs reported to determine the relationship between interval pairs.

Complexity Analysis at P_i $(1 \leq i \leq n)$

Space Complexity of *Log*. at each $P_i, 1 \leq i \leq n$. Each Log at a process stores the start (t^-) and the end (t^+) of an interval. As only one Log entry exists at a time, the space needed at a process P_i at any time is 2 integers.

Space Complexity of Control Messages. sent to P_0 by processes P_1 to P_n.

- As one message is sent per interval, the number of messages is p for each P_i $(i \neq 0)$. This gives a total number of messages as np.
- The size of each message is 2 as each message contains a Log. The total message space overhead for any process is the sum over all the Logs for that process, which is $2p$. Hence the total message space complexity is $2np$.

3 Algorithm *Predicate_Rel*

The algorithm detects a set of intervals, one on each process, such that each pair of intervals satisfies the relationship specified for that pair of processes. If no such set of intervals exists, the algorithm does not return any interval set. The central process P_0 maintains n queues, one for Logs from each process and determines which relation holds between pairs of intervals. The queues are processed using the formalism in [3, 4, 5]. If there exists an interval at the head of each queue and these intervals cannot be pruned, then these intervals satisfy $r_{i,j}\ \forall\ i, j$, where $i \neq j$ and $1 \leq i, j \leq n$. Hence these intervals form a solution set.

We use the *prohibition* function $\mathcal{H}(r_{i,j})$ and the *allows* relation $\rightsquigarrow$ [3, 4, 5, 6]. For each $r_{i,j} \in \Re$, its *prohibition function* $\mathcal{H}(r_{i,j})$ is the set of all relations R such

Table 2. Prohibition functions $\mathcal{H}(r_{i,j})$ for the 13 independent relations $r_{i,j}$ in $\Re$

Relation r	$\mathcal{H}(r_{i,j}(X_i, Y_j))$	$\mathcal{H}(r_{j,i}(Y_j, X_i))$
$p = pb^{-1}$	$\emptyset$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$m = mb^{-1}$	$\{p, m, o, s, f, fb, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$o = ob^{-1}$	$\{p, m, o, s, f, fb, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$s = sb^{-1}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$f = fb^{-1}$	$\{p, m, o, s, f, fb, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$c = cb^{-1}$	$\{p, m, o, s, f, fb, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$q = q^{-1}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$

that if $R(X, Y)$ is *true*, then $r_{i,j}(X, Y')$ can never be *true* for some successor Y' of Y. $\mathcal{H}(r_{i,j})$ is the set of relations that prohibit $r_{i,j}$ from being *true* in the future. Two relations R' and R'' in $\Re$ are related by the *allows* relation $\rightsquigarrow$ if the occurrence of $R'(X, Y)$ does not prohibit $R''(X, Y')$ for some successor Y' of Y.

Definition 1. *Function* $\mathcal{H} : \Re \rightarrow 2^{\Re}$ *is defined to be* $\mathcal{H}(r_{i,j}) = \{R \in \Re \mid if\ R(X, Y)\ is\ true\ then\ r_{i,j}(X, Y')\ is\ false\ for\ all\ Y'\ that\ succeed\ Y\ \}$.

Definition 2. $\rightsquigarrow$ *is a relation on* $\Re \times \Re$ *such that* $R' \rightsquigarrow R''$ *if the following holds. If* $R'(X, Y)$ *is true then* $R''(X, Y')$ *can be true for some* Y' *that succeeds* Y.

Examples
(i) $c \rightsquigarrow o$ because if $c(X, Y)$ is true, then there is a possibility that $o(X, Y')$ is also true, where Y' succeeds Y.
(ii) $m^{-1} \rightsquigarrow f$ because if $m^{-1}(X, Y)$ is true, then there is a possibility that $f(X, Y')$ is also true, where Y' succeeds Y.

Lemma 1. *If* $R \in \mathcal{H}(r_{i,j})$ *then* $R \not\rightsquigarrow r_{i,j}$ *else if* $R \notin \mathcal{H}(r_{i,j})$ *then* $R \rightsquigarrow r_{i,j}$.

Proof. If $R \in \mathcal{H}(r_{i,j})$, using Definition 1, it can be inferred that $r_{i,j}$ is false for all Y' that succeed Y. This does not satisfy Definition 2. Hence $R \not\rightsquigarrow r_{i,j}$. If $R \notin \mathcal{H}(r_{i,j})$, it follows that $r_{i,j}$ can be true for some Y' that succeeds Y. This satisfies Definition 2 and hence $R \rightsquigarrow r_{i,j}$. □

Table 2 gives $\mathcal{H}(r_{i,j})$ for the 13 interaction types in $\Re$. It is constructed by analyzing each interaction pair.

Example: The third row of Table 2 gives the relations o and ob.

- In column two, $\mathcal{H}(o_{i,j}(X_i, Y_j)) = \{p, m, o, s, f, fb, cb, q\}$. Hence, $p(X_i, Y_j)$ or $m(X_i, Y_j)$ or $o(X_i, Y_j)$ or $s(X_i, Y_j)$ or $f(X_i, Y_j)$ or $fb(X_i, Y_j)$ or $cb(X_i, Y_j)$ or $q(X_i, Y_j)$ implies that $o(X_i, Y_j')$ can never hold for any successor Y_j' of Y_j.
- In column three, $\mathcal{H}(ob_{j,i}(Y_j, X_i)) = \{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$. Hence, $p(Y_j, X_i)$ or $m(Y_j, X_i)$ or $mb(Y_j, X_i)$ or $o(Y_j, X_i)$ or $ob(Y_j, X_i)$ or $s(Y_j, X_i)$ or $sb(Y_j, X_i)$ or $f(Y_j, X_i)$ or $fb(Y_j, X_i)$ or $c(Y_j, X_i)$ or $cb(Y_j, X_i)$ or $q(Y_j, X_i)$ implies that $ob(Y_j, X_i')$ can never hold for any successor X_i' of X_i.

The following theorem states that if R' allows R'', then Theorem 1 states that R'^{-1} necessarily does not allow relation R''^{-1}.

Table 3. The "allows" relation $\rightsquigarrow$ on $\Re \times \Re$, in matrix form, to verify Theorem 1

$\rightsquigarrow$	p	pb	m	mb	o	ob	s	sb	f	fb	c	cb	q
p	1												
pb	1	1	1	1	1	1	1	1	1	1	1	1	1
m	1												
mb	1		1		1				1		1		
o	1												
ob	1		1		1				1		1		
s	1												
sb	1		1		1				1		1		
f	1												
fb	1												
c	1		1		1				1		1		
cb	1												
q	1												

Theorem 1. *For $R', R'' \in \Re$ and $R' \neq R''$, if $R' \rightsquigarrow R''$ then $R'^{-1} \not\rightsquigarrow R''^{-1}$.*

The theorem can be observed to be true from Lemma 1 and Table 2 by using a case-by-case analysis. Table 3 shows the grid of the $\rightsquigarrow$ relation for this analysis. A "1" indicates that the row header allows the column header. Alternately, this analysis is easier by using the following form of Theorem 1: "For $R' \neq R''$, if $R' \notin \mathcal{H}(R'')$, then $R'^{-1} \in \mathcal{H}(R''^{-1})$".

(Example 1.) $c \rightsquigarrow o \Rightarrow c^{-1} \not\rightsquigarrow o^{-1}$, which is true.
(Example 2:) $m^{-1} \rightsquigarrow f \Rightarrow m \not\rightsquigarrow f^{-1}$, which is true.

Note $R' \neq R''$ in Theorem 1; otherwise $R' \rightsquigarrow R'$ holds as for p, pb, and c, leading to $R'^{-1} \not\rightsquigarrow R'^{-1}$, a contradiction.

Lemma 2. *If the relationship $R(X, Y)$ between intervals X at P_i and Y at P_j is contained in the set $\mathcal{H}(r_{i,j})$ and $R \neq r_{i,j}$, then X can be removed from the queue Q_i.*

Proof. By definition of $\mathcal{H}(r_{i,j})$, $r_{i,j}(X, Y')$ cannot exist, where Y' is any successor of Y. As $r_{i,j} \neq R$, X cannot be a part of the solution. So X can be deleted. □

Lemma 3. *If the relationship between a pair of intervals X at P_i and Y at P_j is not equal to $r_{i,j}$, then either X or Y is removed from the queue.*

Proof. We use contradiction. Assume relation $R(X, Y)$ $(\neq r_{i,j}(X, Y))$ is true for intervals X and Y. From Lemma 2, the only time neither X nor Y will be deleted is when $R \notin \mathcal{H}(r_{i,j})$ and $R^{-1} \notin \mathcal{H}(r_{j,i})$. From Lemma 1, it can be inferred that $R \rightsquigarrow r_{i,j}$ and $R^{-1} \rightsquigarrow r_{j,i}$. As $r_{i,j}^{-1} = r_{j,i}$, we get $R \rightsquigarrow r_{i,j}$ and $R^{-1} \rightsquigarrow r_{i,j}^{-1}$. This is a contradiction as by Theorem 1, R being unequal to $r_{i,j}$, $R \rightsquigarrow r_{i,j} \Rightarrow$

$R^{-1} \not\rightsquigarrow r_{i,j}^{-1}$. Hence $R \in \mathcal{H}(r_{i,j})$ or $R^{-1} \in \mathcal{H}(r_{j,i})$ or both; so one or both of X and Y can be deleted. □

Lemma 3 guarantees progress; when two intervals are checked, if the desired relationship is not satisfied, at least one of them can be discarded. Further, it is possible that both the intervals being tested are discarded.

Example: We want to detect X and Y, where $r_{i,j}(X, Y) = f$. If $R(X, Y) = o$, we have that $o \not\rightsquigarrow f$; hence $o(X, Y)$ will not allow $f(X, Y')$ to be true for any Y'. Hence X must be deleted. Further, $ob \not\rightsquigarrow fb$ and hence $ob(Y, X)$ will not allow $fb(Y, X')$ to be true for any X'. Hence, Y must also be deleted.

Theorem 2. *Problem Predicate_Rel is solved by the algorithm in Figure 5.*

Proof. The algorithm implements Lemma 2 which allows queues to be pruned correctly. An interval gets deleted only if it cannot be part of the solution. Specifically, interval X gets deleted if $R(X, Y) \in \mathcal{H}(r_{i,j})$ and $R \neq r_{i,j}$ (lines 13,14,17). Similarly, Y is deleted if $R(Y, X) \in \mathcal{H}(r_{j,i})$ and $R \neq r_{i,j}$ (lines 15-17). Further, each interval gets examined unless a solution is found using one of its predecessors. Lemma 3 guarantees that if $R(X, Y) \neq r_{i,j}$, then either interval X or interval Y is deleted. Hence, if *every* queue is non-empty and its head cannot be pruned, then the set of intervals at the head of each queue forms a solution.

The set *updatedQs* stores the indices of all the queues whose heads get updated. In each iteration of the **while** loop, the indices of all the queues whose heads satisfy Lemma 2 are stored in set *newUpdatedQs* (lines (13)-(16)). In lines (17) and (18), the heads of all these queues are deleted and indices of the updated queues are stored in the set *updatedQs*. Observe that only interval pairs which were not compared earlier are compared in subsequent iterations of the **while** loop. The loop runs until no more queues can be updated. If *all* the queues are now non-empty, then a solution is found (Lemma 3), where for the intervals $X = head(Q_i)$ and $Y = head(Q_j)$, $R(X, Y) = r_{i,j}$. □

Theorem 3. *The algorithm in Figure 5 has the following complexities.*

1. *The total message space complexity is* $2np$. *(proved in Section 2)*
2. *The total space complexity at process* P_0 *is* $2np$. *(follows from (1))*
3. *The time complexity at* P_0 *is* $O((n-1)pn))$.

Proof. The time complexity is the product of the number of steps needed to determine a relationship ($O(1)$, follows trivially from Figure 2) and the number of relations determined. For each interval considered from one of the queues in *updatedQs* (lines (6)-(12)), the number of relations determined is $n - 1$. Thus the number of relations determined for each iteration of the **while** loop is $(n - 1)|updatedQs|$. But $\sum |updatedQs|$ over all iterations of the **while** loop is less than the total number of intervals over all the queues. Thus, the total number of relations determined is less than $(n-1) \cdot x$, where $x = pn$ is the upper bound on the total number of intervals over all the queues. As the time required to determine a relationship is $O(1)$, the time complexity is $O((n-1)np)$. □

queue of Log: $Q_1, Q_2, \ldots Q_n = \perp$
set of int: $updatedQs$, $newUpdatedQs = \{\}$
On receiving interval from process P_z at P_0
1: Enqueue the interval onto queue Q_z
2: **if** (number of intervals on Q_z is 1) **then**
3: $updatedQs = \{z\}$
4: **while** ($updatedQs$ is not empty)
5: $newUpdatedQs$={}
6: **for** each $i \in updatedQs$
7: **if** (Q_i is non-empty) **then**
8: X = head of Q_i
9: **for** $j = 1$ to n
10: **if** (Q_j is non-empty) **then**
11: Y = head of Q_j
12: Test for $R(X, Y)$ using interval timestamps (Fig. 2)
13: **if** $(R(X, Y) \in \mathcal{H}(r_{i,j}))$ and $R \neq r_{i,j}$ **then**
14: $newUpdatedQs = \{i\} \cup newUpdatedQs$
15: **if** $(R(Y, X) \in \mathcal{H}(r_{j,i}))$ and $R \neq r_{j,i}$ **then**
16: $newUpdatedQs = \{j\} \cup newUpdatedQs$
17: Delete heads of all Q_k where $k \in newUpdatedQs$
18: $updatedQs = newUpdatedQs$
19 **if** (all queues are non-empty) **then**
20: Heads of queues identify intervals that form the solution.

Fig. 5. On-line algorithm at P_0 to solve $Predicate\_Rel$, based on [4, 5]

4 Conclusions and Discussion

This paper formulated the problem of detecting a global predicate in a (distributed) ubiquitous system assuming the presence of global time. Such ubiquitous systems are becoming common due to the spread of embedded devices that are networked together, sensor networks, and ad-hoc networks. The assumption of an approximate global time axis is also becoming reasonable in an increasing number of scenarios due to the spread and availability of GPS, and inexpensive clock synchronization algorithms. The paper presented an algorithm based on [4, 5] to detect a global predicate specified across the various locations, assuming that event streaming from those locations to a central location is available. This model is reasonable because the dynamic ambient network in the ubiquitous environment connects each device to more powerful processing resources. The proposed algorithm is highly scalable as it has low overhead.

We mention some limitations of the approach. Global time is at best an approximation. A common time axis will not be applicable when predicates based on causality (i.e., happens before) relation [14] are specified. In such cases, the algorithms presented in [3, 4] using the theory in [11, 12] can be used. Also, the availability of global time in some scenarios with limited resources and/or constrained network topologies may not be practical.

The presented formalism assumed *local discreteness* which implied *local interval separation* and no *points*. Variations can be handled by adapting this formalism

(see [13]). For example, if *local interval separation* is relaxed, an interval can begin at the same instant the previous interval at the same process ends. X_i' would be a successor of X_i if $m(X_i, X_i')$ or $p(X_i, X_i')$. In Table 2, $\mathcal{H}(m)$ would exclude f, fb, q, and $\mathcal{H}(s)$, $\mathcal{H}(sb)$, $\mathcal{H}(q)$ would each exclude mb. In Table 3 for $\rightsquigarrow$, there would be "1" for (f, m), (fb, m), (q, m), (mb, s), (mb, sb), (mb, q). Theorem 1 can be seen to still hold. Other variations, such as allowing *points* (one point per clock tick), and about clock properties and time density, can be similarly handled.

References

1. I. Akyildiz, W. Su, Y. Sankarasubramanian, E. Cayirci, Wireless sensor networks: A survey, Computer Networks, 38(4): 393-422, 2002.
2. J. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM, 26(11): 832-843, 1983.
3. P. Chandra, A. D. Kshemkalyani, Detection of orthogonal interval relations, Proc. 9th High-Performance Computing Conference (HiPC), LNCS 2552, Springer, 323-333, 2002.
4. P. Chandra, A. D. Kshemkalyani, Global predicate detection under fine-grained modalities, Proc. ASIAN Computing Conference 2003 (ASIAN), LNCS 2896, Springer, 91-109, Dec. 2003.
5. P. Chandra, A. D. Kshemkalyani, Causality-based predicate detection across space and time, IEEE Transactions on Computers, 54(11): 1438-1453, 2005.
6. P. Chandra, A. D. Kshemkalyani, Global state detection based on peer-to-peer interactions, Proc. IFIP Conf. on Embedded and Ubiquitous Computing (EUC), LNCS, Springer, Dec. 2005.
7. K. M. Chandy, L. Lamport, Distributed snapshots: Determining global states of distributed systems, ACM Trans. Computer Systems, 3(1): 63-75, 1985.
8. J. Elson, K. Romer, Wireless sensor networks: A new regime for time synchronization, First Workshop on Hot Topics In Networks (HotNets-I), October 2002.
9. S. Ganeriwal, R. Kumar, M. Srivastava, Timing-sync protocol for sensor networks, Proc. ACM Conf. Embedded Networked Sensor Systems, 138-149, Nov. 2003.
10. C. L. Hamblin, Instants and intervals, in "The Study of Time," pp. 324-332, Springer-Verlag New York/Berlin, 1972.
11. A. D. Kshemkalyani, Temporal interactions of intervals in distributed systems, Journal of Computer and System Sciences, 52(2): 287-298, April 1996.
12. A. D. Kshemkalyani, A fine-grained modality classification for global predicates, IEEE Trans. Parallel and Distributed Systems, 14(8): 807-816, August 2003.
13. A. D. Kshemkalyani, Predicate detection using event streams in ubiquitous environments, UIC Technical Report UIC-CS-02-05, 2005.
14. L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM, 21(7): 558-565, July 1978.
15. T. Logsdon, The Navstar Global Positioning System, Van Nostrand/Reinhold, New York, 1992.
16. D. Mills, Internet time synchronization: the Network Time Protocol, IEEE Trans. on Communications, 39(10): 1482-1493, October 1991.
17. K. Romer, Time synchronization in ad-hoc networks, Proc. ACM MobiHoc, 2001.
18. B. Sundararaman, U. Buy, A. D. Kshemkalyani, Clock synchronization for wireless sensor networks: A survey, Ad-Hoc Networks, 3(3): 281-323, May 2005.
19. S. Tilak, N. Abu-Ghazaleh, W. Heinzelman, A taxonomy of wireless micro-sensor models, ACM Mobile Computing & Communications Review, 6(2), April 2002.