Querying Context Maps using Relative Timing Predicates in Pervasive Environments

Vaskar Raychoudhury
Deapartment of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, India
vaskar@ieee.org

Ajay D. Kshemkalyani
Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053
ajay@uic.edu

Jiannong Cao
Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, HK
csjcao@comp.polyu.edu.hk

ABSTRACT

Pervasive computing environments are composed of numerous smart entities (objects and human alike) which are interconnected through contextual links in order to create a Web of physical objects. The contextual links can be based on matching context attribute-values (e.g., co-location) or social connections. We call such a Web of smart physical objects a context map. Context maps can be used for context-aware search and browse of the physical world. This paper shows how to evaluate predicates on the context map, when the predicate is specified using complex timing relations.

Categories and Subject Descriptors
H.3.4 [**Context**]: Web View

General Terms
Algorithms

Keywords
Context map; Relative event detection; Searching and browsing physical world; Query; Predicate detection.

1. INTRODUCTION

Rapid advances in embedded sensing technologies, wireless communications, and mobile computing, are transforming our physical world into an intelligent environment. Physical objects (including human beings) embedded with sensing, computing, and communication capabilities are being contextually interconnected to form an Internet of physical objects, not much unlike the traditional Internet. We call such a novel structure a context map where contextual links between pairs of objects are created based on their matching context attributes (e.g., location, ownership, social connections, etc). Context attributes can be static or dynamic depending on whether their value changes with time. Let us consider the following intelligent office example to illustrate the idea.

Example 1. Tom enters his office PQ821 at 9:00 am with a laptop borrowed from the office IT services for presenting at the Annual General Meeting scheduled from 11:00 am. He calls his project partner Bob who arrives at 9:45 am to take a look at his PPT slides. Leaving Bob there Tom goes to the canteen at 10:30 am for breakfast and finally enters meeting room PQ 304 at 10:50 am. He finds that Bob has arrived there at 10:45 am and has set up the laptop for presentation.

There are three smart objects - Tom, Bob and Laptop. All three have a location (Loc) context attribute and the laptop has an additional user attribute. The timing diagram in Fig. 1 shows the change of context attribute values with time and Fig. 2 shows the corresponding contextual links in the context map and the time through which they are active. If necessary, inactive previous links can also be stored to track the past contextual relationships of an object. Like the Web search and browse over the Internet, context map enables users to search for a physical object based on its current context values and to browse through the present and past contextual links between objects. Creation and maintenance of contextual links, however, requires correct and timely detection of contextual events generated by change of values of dynamic context attributes.

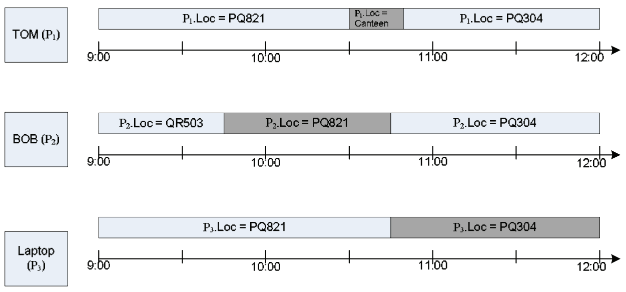

Figure 1. Timing Diagram of Example1

A context map represents a global snapshot of the physical world including multiple smart physical objects and people, and the variations of contextual relations among them with respect to time. A global snapshot should contain one local state from each participating entity. Using a common time axis, a global state can be specified (1) as occurring at the same time instant in each entity (or, concurrent), or (2) in terms of specific temporal relationships among the local states (one local state from each process) (or, relative). Examples 2 and 3 show the concurrent and relative temporal relations, respectively.

Example 2. From Example 1, concurrency of location context of *Tom, Bob and the Laptop* can be represented as (Tom.*Loc* = Bob.*Loc* = Laptop.*Loc*).

Example 3. In Example 1, assume that *after Tom enters his office, he has invited both Bob and Ron to take a look at his slides. After that they may go to the meeting together or separately.* So, here the relative occurrences of these events can be specified as - *Ron*

Midsens'2011, December 12th, 2011, Lisbon, Portugal.

and Bob enter Tom's office AFTER Tom and they leave AT THE SAME TIME or BEFORE Tom.

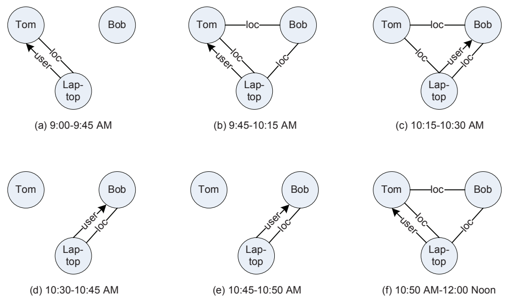

Figure 2. Context Map for Example 1

Incorrect detection of afore-mentioned events will certainly introduce contextual inconsistency in the context map. Moreover, due to the predominantly wireless nature of communication in pervasive computing environments, event reporting by smart objects often suffers from finite but unbounded delays. So, the consistent and timely maintenance of context map in a dynamic and asynchronous pervasive computing environment is non-trivial.

Context consistency detection has been studied in [1][2][3][4] for contexts belonging to the same snapshot of time. On the other hand, [5] has proposed inconsistency detection assuming an inherently asynchronous pervasive computing environment. However, example applications in [5] considered contextual events occurring at the same or close-by locations. This situation does not introduce delay in event reporting and hence, it is not readily evident whether the solution proposed in [5] works effectively for real-life applications. Also, more importantly, none of the above papers can detect contextual events with relative timing relations.

There exists a need to evaluate predicates using complex timing relations on the context map to answer queries such as, "Did Bob and Ron enter Tom's office after him and leave it after him?" None of the existing work can answer queries on the context map based on such complex timing relations. A generic algorithm to answer such complex queries was given in [14]. In this paper, we show how to adapt the algorithm in [14] to detect predicates with relative timing constraints. Although distributed systems are asynchronous, we assume synchronized physical clocks are available within the same administrative domain. Our algorithm leverages the availability of physically synchronized clocks.

Contributions:

1. We introduce the context map to track the image of the physical world, so that queries can be run against the context map.
2. We give instances of real-world problems (Examples 3 and 5) in pervasive environments where complex timing relations are involved in queries on the context map.
3. We show how to adapt the theory and abstract algorithm given in [14] to answer queries based on relative timing predicates in an on-going manner, to a real-world problem using a detailed step-by-step example.

The remainder of the paper is organized as follows. Section 2 discusses the related works. Section 3 gives our system model. Section 4 adapts the algorithm of [14] to handle complex cases of detecting events with relative timing relations. Finally Section 5 concludes this paper with the directions of future work.

2. RELATED WORKS

Contextually connecting smart objects is the key to many novel pervasive computing applications, such as, Internet of Things [6] or Real world search [7]. However, this requires capturing contextual events and relating them based on concurrency of occurrences, or just based on the relative time of their occurrences.

Context maps are studied in [8] and [9] with reference to wireless sensor networks (WSN). A map-based world model has been presented in [8] for WSN where a map is an aggregated view on the spatial and temporal distribution of a certain attribute (e.g., temperature) sensed by some sensor nodes. This approach has limited scope and does not aim to connect all physical objects. SENSID [9] is a situation detecting middleware for WSN which is used to capture spatial and temporal event patterns in WSN using conjunctive situation predicates.

Event detection by specifying predicates is a commonly used policy and is being used in pervasive environments as well. Concurrent events are detected in [5] by tackling temporal inconsistency caused by message asynchrony in pervasive environment. They use a logical clock based approach for detecting concurrent events specified by conjunctive predicates. Later, an extension [10] was made to decide temporal ordering of contextual events generated by a user's activity. However, this process can only detect which event happens before or after the other and cannot find out the relative timing relations between events as introduced in this paper in Example 3.

Predicate detection in traditional distributed computing is an old research area. Detecting distributed predicates based on concurrent and relative timing occurrences of intervals have been studied in [12] and [13], using logical time based [11] causality relationships. An extension of these works is [14], which introduced an approach of detecting composite events with relative timing constraints in WSN, using synchronized clocks.

3. SYSTEM MODEL

We assume that a pervasive computing environment is composed of multiple smart entities connected wirelessly and they communicate through asynchronous message passing. Each entity has a set of context attributes whose values may change with time. We model these changes as the generation of a series of linearly ordered set of discrete events E_i by the execution of a process P_i at each entity. The time duration between two successive events at a process identifies an *interval* during which the value of a context attribute holds (Fig. 1).

Event streams from the processes report intervals to a central data fusion server, P_0, either periodically (in batch mode) or following a trigger-based approach (i.e., as and when the value of an attribute changes). Information about the reported intervals is "fused" at the server and examined to detect relative temporal relations between intervals specified as a global predicate Φ that is satisfied by the current system state. The predicate Φ must be (i) explicitly defined on attribute values during intervals that are (ii) implicitly related using relative timing relationships. The context map is updated based on the truth value of Φ, in order to reflect global states of execution.

Predicates can be of two types: relational and conjunctive predicates. Relational predicates (Example 2) can be true for any values of the context attributes and cannot be evaluated locally. Conjunctive predicates, on the other hand, can be locally evaluated. They must be expressible in conjunctive form, i.e., $\Phi = \Lambda^t_i \Phi_i$, which is a conjunct over the local predicates Φ_i, where timing relations between intervals are included in the conjunction operation Λ^t. We consider queries using only conjunctive predicates. The following example shows a conjunctive predicate version of Example 2.

Example 4. Conjunctive predicate: (Tom.*Loc* = PQ821 & Bob.*Loc* = PQ821 & Laptop.*Loc* = PQ821).

We assume synchronized clocks for all the smart entities. Many low-cost, high-accuracy clock synchronization protocols have been proposed for single and multi-hop wireless sensor networks [18][19][20][21]. The clock skew can be very small (≈ microsecs), relative to the rate of changes in the observed physical phenomena, like human and object movement. Although pervasive environments are asynchronous WSNs, it is reasonable to assume such physically synchronized clocks, particularly if the participating entities belong to the same administrative domain.

4. DETECTING PREDICATES BASED ON RELATIVE TIMING CONSTRAINTS

As already mentioned in Section 1, detecting predicates with relative timing constraints is more challenging. In this section we shall study it in more detail for conjunctive predicates. For a single time axis there are 13 ways in which two time intervals can be related to one another on that time axis [16][17]. For intervals X and Y in processes P_i and P_j, respectively, the 13 relations are illustrated in Fig. 3. The set of these 13 relations is denoted by $\mathfrak{R}$. There are six pairs of inverses as shown and *equals* is its own inverse. Below we define the problem of detecting predicates based on relative timing constraints.

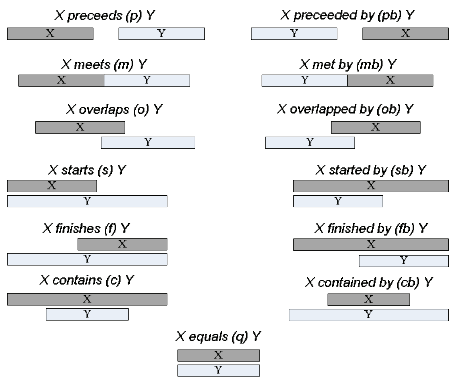

Figure 3. The 13 relations $\mathfrak{R}$ between intervals [16], [17]

4.1 Problem Definition

Problem *Relative*$_{pred}$. Given a set of relations $r^*_{i,j} \subseteq \mathfrak{R}$ for each pair of processes P_i and P_j, determine online the earliest intervals, one from each process, such that any one of the relations in $r^*_{i,j}$ is satisfied (by the intervals) for each (P_i, P_j) pair. If a solution exists, identify the relationship from $\mathfrak{R}$ for each pair of intervals in the solution.

4.2 Data Structure for Relative$_{pred}$

We assume that there is a set of processes (one for each smart entity), P, and each process has a single context attribute A. We also assume that $|P| = p$. Every event (e) is identified by a triple (P_i, Val_A, t_s), where P_i is the identifier of process i, Val_A is the value of attribute, and t_s is the timestamp of occurrence of e.

The central server maintains a queue, called *interval queue* (Q_i), which captures the intervals generated by each process P_i. We assume that this queue can hold at most ξ intervals, where ξ is the maximum number of intervals per process.

4.3 Explanation

Consider Example 3 which specifies a predicate using three different relative timing constraints - AFTER, AT THE SAME TIME and BEFORE. Assuming Ron, Tom and Bob as three processes, P_i, P_j, and P_k, generating intervals X, Y and Z, respectively, the timing relations of Example 3 are shown in Fig. 4. We can easily see that the timing relations between a pair of intervals are not fixed or unique and can be any one from a subset of the 13 relations shown in Fig. 3 and the predicate is satisfied if any one type of relation holds.

So, for X, Y, and Z the subset of timing relations can be represented as $r^*_{i,j}=\{cb, f\}$, $r^*_{i,k}=\{sb, s, q\}$, and $r^*_{j,k}=\{c, fb\}$.

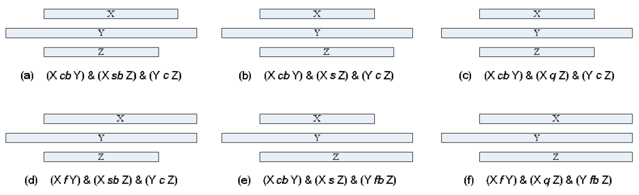
(a) (X cb Y) & (X sb Z) & (Y c Z)
(b) (X cb Y) & (X s Z) & (Y c Z)
(c) (X cb Y) & (X q Z) & (Y c Z)
(d) (X f Y) & (X sb Z) & (Y c Z)
(e) (X cb Y) & (X s Z) & (Y fb Z)
(f) (X f Y) & (X q Z) & (Y fb Z)

Figure 4. Timing Relations for Example 3

Let us consider another example of detecting predicates with relative occurrences of time intervals where contextual events have occurred at physically distant locations.

Example 5. *ABC Logistics Company has a secure datacenter (DC). Accessing data from the DC requires presentation of fingerprints by the DC manager (DCM), and two deputy general managers (DGM) of the company. All three managers are distributed across the globe and they must present their fingerprint remotely, over the wire. The DCM first presents his fingerprint to activate the system AFTER which the two DGMs present their fingerprints. They hold their fingers until the fingerprints are verified by the DCM and an acknowledgement is sent. AFTER both the DGMs pass security verifications, the DCM removes his finger and the DC becomes accessible to the user.*

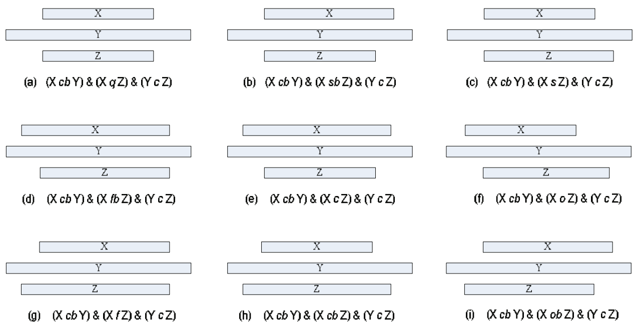
(a) (X cb Y) & (X q Z) & (Y c Z)
(b) (X cb Y) & (X sb Z) & (Y c Z)
(c) (X cb Y) & (X s Z) & (Y c Z)
(d) (X cb Y) & (X fb Z) & (Y c Z)
(e) (X cb Y) & (X c Z) & (Y c Z)
(f) (X cb Y) & (X o Z) & (Y c Z)
(g) (X cb Y) & (X f Z) & (Y c Z)
(h) (X cb Y) & (X cb Z) & (Y c Z)
(i) (X cb Y) & (X ob Z) & (Y c Z)

Figure 5. Timing Relations for Example 5

Fig. 5 describes the timing relations between the intervals for this example (assuming DCM as Y and the two DGMs as X and Z, respectively). So, for X, Y, and Z the subset of timing relations can be represented as $r^*_{i,j}=\{cb\}$, $r^*_{i,k}=\{sb, s, fb, f, o, ob, c, cb, q\}$, and $r^*_{j,k}=\{c\}$.

In *Relative*$_{pred}$ problem, for each pair of processes (P_i, P_j), there is a set $r^*_{i,j} \subseteq \mathfrak{R}$ such that some relation in $r^*_{i,j}$ must hold. For solving *Relative*$_{pred}$, given an arbitrary $r^*_{i,j}$, the central server stores all the incoming intervals in specific interval queues and compares every possible pair of intervals in order to detect a relation. This process tends to grow exponentially unless somehow the numbers of comparisons are restricted by controlling the growth of the interval queues. So, to avoid tracking multiple intervals in each queue and examining exponential number of global states (up to p^n), we use the *prohibition function* and *CONVEXITY property* from [14].

4.4 Algorithm for Solving Relative$_{pred}$ Problem

To solve the *Relative*$_{pred}$ problem for Example 5, we show how to adapt the theory and abstract algorithm given in [14]. As already mentioned, before giving the algorithm, we briefly describe the prohibition function and *CONVEXITY* property and introduce certain lemmas derived from them that have been used in this paper.

Table 1. Prohibition Functions H($r_{i,j}$) for the 13 Relations $r_{i,j}$ in $\mathfrak{R}$

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

Definition 1. *Prohibition function* H: $\mathfrak{R} \rightarrow 2^{\mathfrak{R}}$ *is*

H($r_{i,j}$) = {$R \in \mathfrak{R}$ | *if* $R(X, Y)$ *is true then* $r_{i,j}(X, Y')$ *is false for all* Y' *that succeed* Y}

Intuitively, for each $r_{i,j} \in \mathfrak{R}$, the prohibition function H($r_{i,j}$) is the set of all orthogonal relations R ($\neq r_{i,j}$) such that if $R(X, Y)$ is true, then $r_{i,j}(X, Y')$ can never be true for any successor Y' of Y. H($r_{i,j}$) is the set of relations that prohibit $r_{i,j}$ from being true in the future.

Table 1 gives H($r_{i,j}$) for the 13 relations in $\mathfrak{R}$. It is constructed by analyzing each relation pair. Example 6 shows the use of prohibition function in pruning interval queues.

Example 6. The sixth row of Table 1 gives H($r_{i,j}$) for the relations c and cb.

- In column two, H($c_{i,j}(X_i, Y_j)$) ={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 $c(X_i, Y'_j)$ can never hold for any successor Y'_j of Y_j.
- In column three, H($cb_{j,i}(Y_j, X_i)$) = {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 $cb(Y_j, X'_i)$ can never hold for any successor X'_i of X_i.

The following two lemmas directly follow from the prohibition function. Lemma 2 guarantees progress, when two intervals are checked, if the desired relationship is not satisfied, at least one of them can be discarded.

Lemma 1. *If the relationship R(X, Y) between intervals X at* P_i *and Y at* P_j *is contained in the set* H($r_{i,j}$) *and* $R \neq r_{i,j}$, *then X can be removed from the queue* Q_i.

Lemma 2. *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.*

So far, we specified a single relation r from $\mathfrak{R}$ between a pair of intervals. Queries based on more complex timing relations, e.g., Figure 5 for Example 3, and Figure 6 for Example 5, show that r can be any one of several relations in a subset of $\mathfrak{R}$. We now need to identify the counterpart of Lemma 2. To better illustrate this issue, let us reconsider Fig. 4, where we see that for Example 3, $r^*_{j,k}=\{c, fb\}$. The intervals considered are Y and Z, and let us assume, that a relation (other than c or fb) '*precedes*' $p(Y, Z)$, and hence, $pb(Z, Y)$ holds between them. We want to check whether it is possible to remove either of the intervals, Y and Z. This is illustrated in Fig. 6.

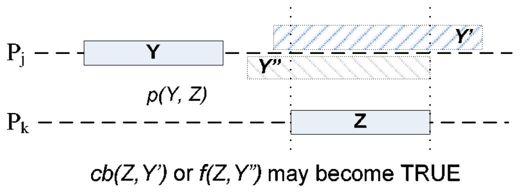

Figure 6. Example of "no queue pruning" of Q_k (for P_k)

- We can see that $p \in$ H(c), but $pb \notin$ H(cb). Even though Y will not form a part of a solution satisfying relation c with any future Z', Z can form a solution satisfying relation cb with any future Y', i.e., $cb(Z, Y')$ may be true. So, the record of Z must be retained in Q_k.
- Similarly, for the other relation fb, $p \in$ H(fb), $pb \notin$ H(f). Even though Y will not form a part of a solution satisfying relations fb with any future Z', Z can form a solution satisfying relation f with any future Y', So, as before, $f(Z, Y')$ may be true for some future Y', So, the record of Z must be retained in Q_k.

So, in neither case, Q_k can be pruned, i.e., records of Z cannot be deleted from its interval queue. Next we define the CONVEXITY property [14] on $r^*_{i,j}$.

Definition 2. CONVEXITY:

$$\forall R \notin r^*_{i,j} : (\forall r_{i,j} \in r^*_{i,j}, R \in \mathrm{H}(r_{i,j}) \vee \forall r_{j,i} \in r^*_{j,i}, R^{-1} \in \mathrm{H}(r_{j,i}))$$

It guarantees that either X or Y or both get deleted if $R(X, Y) \notin r^*_{i,j}$ and, hence, there is no explosion of global states. The following results from [14] are used.

Lemma 3. *If the relation R(X, Y) between intervals X and Y (at processes* P_i *and* P_j, *respectively) is contained in the set* $\bigcap_{r_{i,j} \in r^*_{i,j}}$ H($r_{i,j}$), *then interval X can be removed from* Q_i.

Lemma 4. *If the relation R(X, Y) between a pair of intervals X and Y (at processes* P_i *and* P_j, *respectively) does not belong to the set* $r^*_{i,j}$, *where* $r^*_{i,j}$ *satisfies CONVEXITY, then either interval X or Y is removed from the queue.*

From the above lemmas we can say that if a $r^*_{i,j}$ can satisfy CONVEXITY, it can be solved and our algorithm (given later) can be used to detect *Relative*$_{pred}$. We apply the lemmas on the $r^*_{i,j}$ of Example 5 and verify that it satisfies CONVEXITY. We proceed as follows:

1. We know that Example 5 have the sets of relations between the intervals (*X*, *Y* and *Z*), $r^*_{i,j} = \{cb\}$, $r^*_{j,k} = \{c\}$, and $r^*_{i,k} = \{o, ob, s, sb, f, fb, c, cb, q\}$.
2. For each R in $\Re \setminus r^*_{i,j} = \{p, pb, m, mb, o, ob, s, sb, f, fb, c, q\}$, observe from Table 1 that $R \in H(\theta)$ or $R^{-1} \in H(\theta^{-1})$, where $\theta \in r^*_{i,j}$. Similarly, it is true for the other two sets, $r^*_{j,k}$ and $r^*_{i,k}$. So, CONVEXITY is satisfied by $r^*_{i,j}$, $r^*_{j,k}$ and $r^*_{i,k}$ and our algorithm can be used.

Algorithm: Online Algorithm for *Relative*$_{pred}$

```
Interval: ($Val$, $t_s$, $t_f$)
Number of processes: $n = 3 = \{i, j, k\}$
Queue of intervals: $Q_i, Q_j, Q_k = (default, t, \perp)$
Set of integer: $updatedQs$, $newupdatedQs = \{\}$
On receiving interval from process $P_\alpha$ at $P_0$
(1)  Enqueue the interval onto $Q_\alpha$
(2)  if ($|Q_\alpha| = 1$) then
(3)    $updatedQs = \{P_\alpha\}$
(4)    while ($updatedQs \neq$ NULL)
(5)      $newupdatedQs = \{\}$
(6)      for each $\beta \in updatedQs$
(7)        if($Q_\beta \neq$ NULL) then
(8)          $U$ = head of $Q_\beta$
(9)          for $\gamma$ in $\{i, j, k\}$, ($\gamma \neq \beta$)
(10)           if ($Q_\gamma \neq$ NULL) then
(11)             $V$ = head of $Q_\gamma$
(12)             case
//Test for $R(U,V)$ using interval timestamps (Fig.5)
(13)             $(\beta, \gamma) = (i, j)$ or $(j, i)$:
(14)               if ($R(U, V) \in \bigcap_{r_{i,j}\in r^*_{i,j}} H(r_{i,j})$ and $R \notin r^*_{i,j}$) then
(15)                 $newupdatedQs = \{i\} \cup newupdatedQs$
(16)               if ($R(V, U) \in \bigcap_{r_{j,i}\in r^*_{j,i}} H(r_{j,i})$ and $R \notin r^*_{j,i}$) then
(17)                 $newupdatedQs = \{j\} \cup newupdatedQs$
(18)             $(\beta, \gamma) = (i, k)$ or $(k, i)$:
(19)               if ($R(U, V) \in \bigcap_{r_{i,k}\in r^*_{i,k}} H(r_{i,k})$ and $R \notin r^*_{i,k}$) then
(20)                 $newupdatedQs = \{i\} \cup newupdatedQs$
(21)               if ($R(V, U) \in \bigcap_{r_{k,i}\in r^*_{k,i}} H(r_{k,i})$ and $R \notin r^*_{k,i}$) then
(22)                 $newupdatedQs = \{k\} \cup newupdatedQs$
(23)             $(\beta, \gamma) = (j, k)$ or $(k, j)$:
(24)               if ($R(U, V) \in \bigcap_{r_{j,k}\in r^*_{j,k}} H(r_{j,k})$ and $R \notin r^*_{j,k}$) then
(25)                 $newupdatedQs = \{j\} \cup newupdatedQs$
(26)               if ($R(V, U) \in \bigcap_{r_{k,j}\in r^*_{k,j}} H(r_{k,j})$ and $R \notin r^*_{k,j}$) then
(27)                 $newupdatedQs = \{k\} \cup newupdatedQs$
(28)     Delete heads of all $Q_t$ where $t \in newupdatedQs$
(29)     $updatedQs = newupdatedQs$
(30)   if (all queues are non-empty) then
(31)     Heads of queues identify intervals that form the solution
```

Above, we present our algorithm to solve the *Relative*$_{pred}$ problem. The algorithm is written in a way to solve an instance of the Example 5 described earlier. As we already know that the set of relations between every pair of processes in Example 5 satisfies CONVEXITY, it is possible to detect the specified relative timing relations between the same processes using the above algorithm.

In the above algorithm, we can replace some expressions with the following values. From Table 1, we have:

For $r^*_{i,j} = \{cb\}$, the set

$$\bigcap_{r_{i,j}\in r^*_{i,j}} H(r_{i,j}) = \{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}\ldots(1)$$

$$\bigcap_{r_{i,j}\in r^*_{i,j}} H(r_{i,j}) \setminus r^*_{i,j} = \{p, m, mb, o, ob, s, sb, f, fb, c, q\}\ldots(2)$$

For $r^*_{j,i} = \{c\}$, the set

$$\bigcap_{r_{j,i}\in r^*_{j,i}} H(r_{j,i}) = \{ p, m, o, s, f, fb, cb, q \}\ldots(1')$$

$$\bigcap_{r_{j,i}\in r^*_{j,i}} H(r_{j,i}) \setminus r^*_{j,i} = \{ p, m, o, s, f, fb, cb, q \}\ldots(2')$$

For $r^*_{i,k} = \{o, ob, s, sb, f, fb, c, cb, q\}$, the set

$$\bigcap_{r_{i,k}\in r^*_{i,k}} H(r_{i,k}) = \{p, m, o, s, f, fb, cb, q\} \ldots\ldots (3)$$

$$\bigcap_{r_{i,k}\in r^*_{i,k}} H(r_{i,k}) \setminus r^*_{i,k} = \{p, m\} \ldots\ldots (4)$$

For $r^*_{k,i} = \{o, ob, s, sb, f, fb, c, cb, q\}$, the set

$$\bigcap_{r_{k,i}\in r^*_{k,i}} H(r_{k,i}) = \{p, m, o, s, f, fb, cb, q\} \ldots\ldots (3')$$

$$\bigcap_{r_{k,i}\in r^*_{k,i}} H(r_{k,i}) \setminus r^*_{k,i} = \{p, m\} \ldots\ldots (4')$$

For $r^*_{j,k} = \{ c \}$, the set

$$\bigcap_{r_{j,k}\in r^*_{j,k}} H(r_{j,k}) = \{p, m, o, s, f, fb, cb, q\} \ldots\ldots(5)$$

$$\bigcap_{r_{j,k}\in r^*_{j,k}} H(r_{j,k}) \setminus r^*_{j,k} = \{ p, m, o, s, f, fb, cb, q \}\ldots (6)$$

For $r^*_{k,j} = \{ cb \}$, the set

$$\bigcap_{r_{k,j}\in r^*_{k,j}} H(r_{k,j}) = \{ p, m, mb, o, ob, s, sb, f, fb, c, cb, q \} \ldots(5')$$

$$\bigcap_{r_{k,j}\in r^*_{k,j}} H(r_{k,j}) \setminus r^*_{k,j} = \{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}..(6')$$

Substituting the values in lines 14-17, 19-22, and 24-27 of Algorithm gives the code in the following Fig. 7.

Analysis of worst case time complexity: The time complexity is the product of the number of steps needed to determine a relationship 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 (p-1). Thus, number of relations determined for each iteration of the **while** loop is (p-1)|*updatedQs*|. But, Σ|*updatedQs*| over all iterations of the **while** loop is less than the total number of intervals over all of the queues. Thus, the total number of relations determined is less than (p-1).X, where $X = p.\xi$ is the upper bound on the total number of intervals over all the queues. As the time required to determine a relationship is $O(1)$ (follows trivially from Fig. 4), the total time complexity is $O(p^2\xi)$.

```
(14)    if $R(U, V) \in \{p, m, mb, o, ob, s, sb, f, fb, c, q\}$ then
(15)        $newupdatedQs = \{i\} \cup newupdatedQs$
(16)  if $R(V, U) \in \{p, m, o, s, f, fb, cb, q\}$ then
(17)        $newupdatedQs = \{j\} \cup newupdatedQs$
(19)    if $R(U, V) \in \{p, m\}$ then
(20)        $newupdatedQs = \{i\} \cup newupdatedQs$
(21)    if $R(V, U) \in \{p, m\}$ then
(22)        $newupdatedQs = \{k\} \cup newupdatedQs$
(24)    if $R(U, V) \in \{p, m, o, s, f, fb, cb, q\}$ then
(25)        $newupdatedQs = \{j\} \cup newupdatedQs$
(26)    if $R(V, U) \in \{p, m, mb, o, ob, s, sb, f, fb, c, q\}$ then
(27)        $newupdatedQs = \{k\} \cup newupdatedQs$
```

Figure 7. Algorithm *Relative*$_{pred}$ replacing for Example 5

5. CONCLUSION AND FUTURE WORKS

In this paper we presented a context map structure which is a graph showing contextual interconnection between several smart physical objects and people of our surrounding environment. Contextual links are added between smart objects based on multiple context-based relations, such as, *co-location* ('located in the same room'), or *colleague* ('employed by the same company'), etc. Since, the dynamic context attributes (e.g., location) of an object may change with time, the contextual links may also be created or deleted with changes in context values. Tracking these changes and answering queries on the context map in a timely and consistent manner is non-trivial in asynchronous pervasive computing environments. We showed how to adapt an existing theoretical algorithm for detecting contextual events with relative timing relations in a real-life example in a pervasive environment. Our proposed algorithm is applicable to a wide range of pervasive applications.

In future, we want to develop decentralized algorithm for detecting contextual events which can maintain context map through autonomous coordination of smart objects. We also want to carry out more experiments to verify our algorithms.

6. REFERENCES

[1] Xu, C. and Cheung, S.C. 2005. Inconsistency detection and resolution for context-aware middleware support. In *Proceedings of the ACM SIGSOFT Int'l Symposium on Foundations of Software Engineerin*g (FSE), pp.336-45.

[2] Xu, C. and Cheung, S.C., and Chan, W.K. 2006. Incremental consistency checking for pervasive context. In *Proceedings of the Int'l Conf. on Software Engineering* (ICSE'06), Shanghai, China, pp. 292–301.

[3] Bu, Y., Gu, T., Tao, X., Li, J., Chen, S., and Lu, J. 2006. Managing quality of context in pervasive computing. In *Proceedings of the Int'l Conf. on Quality Software* (QSIC'06), Beijing, China, pp. 193–200.

[4] Bu, Y., Chen, S., Li, J., Tao, X., and Lu, J. 2006. Context consistency management using ontology based model. In *Proceedings of the Current Trends in Database Technology* (EDBT), Munich, pp. 741–755.

[5] Huang, Y., Ma, X., Cao, J., Tao, X., and Lu, J. 2009. Concurrent Event Detection for Asynchronous consistency checking of pervasive context. In *Proceedings of the IEEE Int'l Conf. on Pervasive Computing and Communications*.

[6] Gershenfeld, N., Krikorian, R., and Cohen, D. 2004. The Internet of Things. *Scientific American*, vol. 291, pp. 76-81.

[7] Wang, H., Tan, C.C., and Li, Q. 2010. Snoogle: A Search Engine for Pervasive Environments," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 8, pp.1188-1202.

[8] Khelil, A., Sheikh, F.K., Ayari, B., and Suri, N. 2008. MWM: A Map-based World Model for Event-driven Wireless Sensor Networks. In *Proceedings of the 2nd ACM International Conference on Autonomic Computing and Communication Systems* (AUTONOMICS).

[9] Kranz, M. 2005. SENSID: A situation detector for sensor networks. *Honours Thesis*, School of Computer Science and Software Engineering, University of Western Australia.

[10] Huang, Y., Ma, X., Cao, J., Tao, X., and Lu, J. Checking Behavioral Consistency Constraints for Pervasive Context in Asynchronous Environments. arXiv:0911.0136

[11] Lamport, L. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, vol. 21, no. 7, pp.558-65.

[12] Chandra, P., Kshemkalyani, A.D. 2005. Causality-based Predicate Detection across Space and Time. *IEEE Trans. on Computers*, 54(11): 1438-1453, (November 2005).

[13] Kshemkalyani, A.D. 2003. A Fine-Grained Modality Classification for Global Predicates. *IEEE Trans. on Parallel and Distributed Systems*, vol. 14, no. 8, pp. 807-816.

[14] Kshemkalyani, A.D. 2007. Temporal Predicate Detection Using Synchronized Clocks. *IEEE Trans. on Computers*, vol. 56, no. 11, pp. 1578-1584, (June 2007).

[15] Mayo, J. and Kearns, P. 1995. Global predicates in rough real time. In *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*.

[16] Allen, J. 1983. Maintaining Knowledge about Temporal Intervals. *Comm. ACM*, vol. 26, no. 11, pp. 832-843.

[17] Hamblin, C.L. 1972. Instants and Intervals The Study of Time, pp. 324-332. Springer-Verlag.

[18] Sichitiu, M.L. and Veerarittiphan, C. 2003. Simple, accurate time synchronization for wireless sensor networks. *IEEE Wireless Communications and Networking* (WCNC).

[19] Kyoung-lae, N., Serpedin, E., and Qaraqe, K. 2008. A New Approach for Time Synchronization in Wireless Sensor Networks: Pairwise Broadcast Synchronization. *IEEE Trans. Wireless Communications*, vol. 7, no. 9, pp.3318-3322.

[20] Sundararaman, B., Buy, U., and Kshemkalyani, A.D. 2005. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, vol.3, Issue 3, Pages 281-323.

[21] Su, W., and Akyildiz, I. 2005. Time-Diffusion Synchronization Protocol for Sensor Networks. *IEEE/ACM Trans. Networking*, vol. 13, no. 2, pp. 384-397.