Context Map for Navigating the Physical World

Vaskar Raychoudhury
Dept. of E&CE, IIT Roorkee, India & Dept. of Computing, Hong Kong Polytechnic University
vaskar@ieee.org

Jiannong Cao
Dept. of Computing, Hong Kong Polytechnic University, Hong Kong
csjcao@comp.polyu.edu.hk

Weiping Zhu
Dept. of Computing, Hong Kong Polytechnic University, Hong Kong
csweizhu@comp.polyu.edu.hk

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

***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 as *context map*. Context maps can be used for context-aware search and browse of the physical world. However, changes of dynamic context values over time may render a context map inconsistent. So, it is important to update contextual links with changes in specific context values. Given the asynchronous nature of pervasive environments, it is non-trivial to detect events generated by contextual changes in real time. We propose two algorithms for instantaneous and periodic detection of events with concurrent timing relations. Our algorithms have low time complexity and they can address the needs of different types of pervasive computing applications. We have evaluated our proposed algorithms through simulations as well as testbed experiments.**

Keywords- Context map; Concurrent event detection; Searching and browsing physical world.

I. INTRODUCTION

Rapid developments in embedded sensing technologies, wireless communications, and mobile computing, are transforming our physical world into a smart space. Physical objects (including people) 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 contextual 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 setup 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.

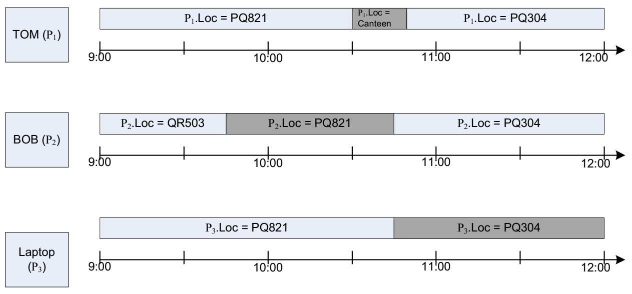

Figure 1. Timing Diagram of Example 1

A context map represents 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 as occurring at the same time instant in each entity (or, concurrent). Example 2 shows the concurrent temporal relations.

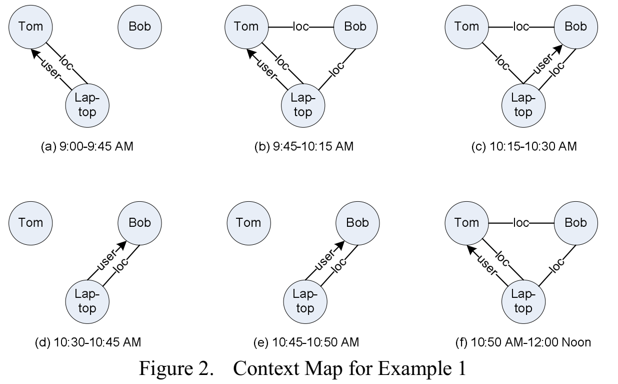

Figure 2. Context Map for Example 1

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*).

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 suffer 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 solution proposed in [5] works effectively for real-life applications.

In this paper, we classify pervasive computing applications based on event reporting delay and the event processing interval at the central server and provide case by case solutions for each of them. Depending on the instantaneous or periodic detection of concurrent events at the central server, we propose two online centralized algorithms.

The remainder of the paper is organized as follows. Section II discusses the related works. Section III gives our system model and classifies pervasive computing applications based on their requirements for event detection. Section IV presents two online algorithms for concurrent event detection. Section V presents our simulations experiments whereas Section VI describes the testbed deployment of a context map used in a smart logistics application. Finally, Section VII concludes this paper with the directions of future work.

II. 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.

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. In [11], algorithms are given to immediately detect conjunctive and relational predicates when they become true.

Predicate detection in traditional distributed computing is an old research area. Detecting distributed predicates based on concurrent timing occurrences of intervals have been studied in [13], using logical time based [12] causality relationships. Detecting predicates based on relative timing constraints have been studied by us in [14].

III. System Model and Event Detection Techniques

In this section we describe our system model and provide a classification of different event detection techniques.

A. 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).

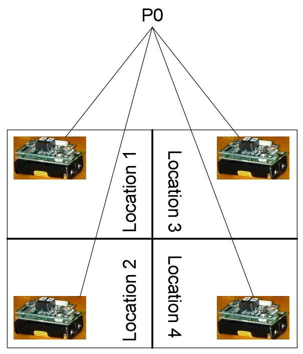

Figure 3. Different Sensors Sense Different Locations (event reporting delay is directly proportional to the distances of the locations from P_0)

Event streams from the processes report intervals to a central data fusion server (Fig. 3), 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 the concurrent 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.

We consider both 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. The following example shows a conjunctive predicate version of Example 2.

Example 3. 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]. The clock skew can be very small (≈ microsecs), relative to the rate of changes in the observed physical phenomena, like human and object movement.

B. *Classification of Event Detection Techniques*

Reporting local events to the central server by a smart entity incurs some message transmission delay (τ). If τ is negligible with respect to the time between two successive predicate evaluations at the server, then the event reporting is called *instantaneous*, otherwise it is *asynchronous*. Instantaneous reporting is feasible for small and bounded area, like home or office environment. Wide open areas, on the other hand, require considerable event reporting delays, as sensors are distributed across faraway locations, e.g., tsunami detection systems where sensory data from ground level and sea beds are combined, or wild-life monitoring in dense forests. However, in most pervasive computing applications, there is an inherent asynchrony caused by message sending delay at the sensors, message propagation delay, handling delay at the server, etc. Addressing the challenges associated with the asynchrony in event reporting is necessary for correct detection of Φ.

TABLE I. CLASSIFICATION OF EVENT DETECTION TECHNIQUES

		Event Reporting Delay	
		Asynchronous (bounded by Δ)	***Instantaneous (Δ = 0)***
Predicate Evaluation	***Trigger-based***	Highway accident detection, Damage detection in long distance oil pipelines, Undersea cables, etc	Safety-critical applications, (Air or Nuclear accident detection, Tsunami detection), Smart homes, office, etc
	Periodic (Batch)	Wild-life / Habitat / Volcano monitoring	Structure health monitoring

Moreover, depending on the type of pervasive application, the server may choose to do predicate checking either periodically or using a trigger-based approach. In the periodic approach, the server stores the incoming events in a buffer and evaluates them periodically. The period of evaluation is called *epoch*. For trigger-based approach, whenever the value of an attribute changes in a process, it reports the event to the server for immediate processing. Depending on the event reporting delay and the predicate evaluation techniques at the server, four different cases can be specified. Table I summarizes the cases with examples.

Here, we assume that the upper bound on event reporting delays is Δ. Since, temporal relations between event intervals are specified with predicates, we shall use event and predicate detection interchangeably for the rest of the paper. We propose two different online algorithms to address trigger-based and periodic detection of concurrent predicates considering asynchronous event reporting (see Section IV). In case $\Delta = 0$, the event reporting is instantaneous and our algorithms can equally handle that. So, our proposed algorithms can address concurrent predicate detection for a wide array of pervasive computing applications hitherto unaccomplished by any other scheme.

IV. CONTEXT MAP CREATION AND CONCURRENT PREDICATE DETECTION

In this section, we first define the concurrent predicate detection problem ($Conc_{pred}$) and then present two algorithms for detecting concurrent predicates and creating context map.

Problem $Conc_{pred}$. Given a set of processes P, such that, each process has a set of k attributes, $A = \{A_1, A_2, \ldots, A_k\}$, each attribute can take up any value from a value set for the attribute, and the value of an attribute may change over time. Assume that a predicate Φ is specified over ($P_i.a_j$, $\forall P_i \in P$ $\forall a_j \in A$). The objective is to identify in an online manner each set of intervals $\mathcal{I} = \{I_1; I_2; \ldots I_n\}$, where I_i is from process P_i, such that there is some instant of time that belongs within all these intervals at which Φ is true.

Events sent by different processes are checked pairwise at the central server and when concurrent predicates are satisfied for a pair of entities, a link is added between them in the context map. So, context map creation and concurrent predicate detection are carried out simultaneously in our proposed algorithms.

A. *Data Structure for $Conc_{pred}$ Algorithms*

We assume that there is a set of processes (one for each smart entity), P, and each process has a set, A, of context attributes. We also assume that $|P| = p$ and $|A| = a$. Every event (e) is identified by a quadruple (P_i, A_j, Val, t_s), where P_i is the identifier of process i, A_j is the attribute j of P_i, Val is the value of attribute A_j, and t_s is the timestamp of occurrence of e. From Fig. 1, when Bob's location changes from QR503 to PQ821, a new event is generated which is represented as (P_1, *Loc*, *PQ821*, *9:45 am*). Similarly, every interval ($\mathcal{I}$) is identified by a triple (Val, t_s, t_f), where Val denotes the value of a context attribute during interval $\mathcal{I}$, which started at time t_s and continued till t_f. The interval started by Bob's location change is represented as (*PQ821*, *9:45 am*, *10:45 am*). A contextual link is represented with a quadruple (A_j, Val, t_s, t_f), where A_j is a context attribute and Val is the value of A_j during the time interval which started at time t_s and continued till t_f. Contextual links are created between a pair of processes iff a context attribute of one process is related to a context attribute (matching or not) of the other process through some user define function, f. E.g., *co-owned(U.owner=V.owner)*, means if the *owner* attributes of U and V processes are same, then they satisfy the *co-owned* function. Another type is *ownership(U.owner = V.id)*, which means that if the *owner* attribute of process U contains the value of *id* attribute of process V, then U is owned by V and they satisfy the *ownership* function.

Two different queues are maintained at the central server. One single *queue of events* (Q) holds a list of incoming events sorted with respect to t_s. Another set of $p*a$ queues, called *interval queues* ($Q\,[i, j]$), are maintained to capture the intervals generated by each attribute of each process. We assume that each such queue can hold at most ξ intervals, where ξ is the maximum number of intervals per attribute per process (for trigger-based predicate detection) or per attribute per epoch (for periodic predicate detection).

B. Algorithm for Trigger-based $Conc_{pred}$ Detection

When a process identifies a change in value of a context attribute, it generates an event and sends it to the server. After a new event arrives at the server, it enqueues the event in Q and starts a timer for time Δ to capture all other events which occurred within (t_s–Δ, t_s) and is delayed during transmission. When the timer expires, the server transfers the event from the head of Q to the head of $Q\,[i, j]$, removing the previous head element of $Q\,[i, j]$. So, the interval for the previous event is closed and a new interval is started for the attribute A_j of process P_i, and it will continue until the attribute value changes to generate a new event. Thus $Q\,[i, j]$ always has at most one element at any time for all i and j.

After a new interval is started at a $Q\,[i, j]$, the attribute values of the intervals at the heads of all $Q\,[i, j]$ are evaluated to check (i) whether any pair has matching attribute-values in which case a contextual link is added, and (ii) whether the predicate Φ is satisfied. Below, we shall elaborate the process using the Example 1.

Algorithm1: Online Algorithm for Trigger-based $Conc_{pred}$

```
Event: ($P_i$, $A_j$, $Val$, $t_s$)
Interval: ($Val$, $t_s$, $t_f$)
Initialize:
    queue of events: $Q = <\perp>$
    queue of intervals: $(\forall p\ \ \forall a)\ Q\,[p, a] \leftarrow enqueue\,(default, t, \perp)$
On receiving an event e from process $P_i$ at $P_0$ due to change of attribute $A_i$
(1) Enqueue $e = (P_i, A_j, Val, t_s)$ in sorted $Q$ and start timer for $(t_s+\Delta)$
When timer pops at $t'$
(2) $e = (P_i, A_j, Val, t_s) \leftarrow$ Dequeue ($Q$)
(3) Dequeue ($Q\,[i, j]$)
(4) Enqueue (($Val$, $t_s$, $\perp$), $Q\,[i, j]$))
(5) for all $a$ in $A$ do
(6)     $\forall m,n \in P$
(7)         if($f(m.a = n.a)$)
(8)             add a link $L(A_j, Val, t_s, \perp)$ in the context map between $m$, $n$
(9)         else
(10)            close any existing link with current time stamp: $L(A_j, Val, t_s, t_f)$
between $m$, $n$
(11) if $\Phi$ (($\forall p\ \ \forall a$) head ($Q\,[p, a]$).$Val$) = TRUE then
(12)    set ALARM
```

From Fig. 4, when a new interval is started when $P_2.Loc$ changes from QR503 to PQ821, a new event is sent by process P_2 and comparison is done between the location attribute of the three entities, Tom, Bob, and Laptop and contextual links are created between them (Fig. 2(b) and 2(c)). At the same time, the relational predicate shown in Example 2 and the conjunctive predicate shown in Example 3 are also satisfied. Similarly, when $P_1.Loc$ changes from PQ821 to canteen, new event is sent by P_1 to the server and the context map is updated to the new state as shown in Fig. 2(d). Again, at 10:45 am, two events are concurrently generated at P_2 and P_3 (both $P_2.Loc$ and $P_3.Loc$ change from PQ821 to PQ304) and they are detected by the Algorithm1 and updated in the context map (Fig. 2(e)). Finally, $P_1.Loc$ changes from canteen to PQ304 and concurrency is detected between the location attributes of the three process and the context map is updated to the one in Fig. 2(f). In this case also, the predicates specified in Example 2 and Example 3 are satisfied. Context maps can store the old contextual links to track the past locations and users of the laptop ("where the laptop was at 10:35 am and who was using it?").

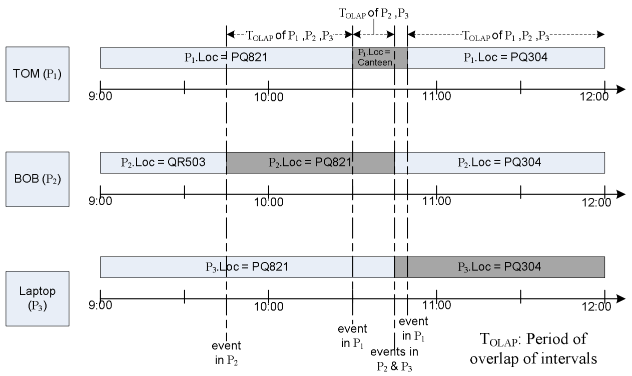

Figure 4. Generation of Events in Example 1

Analysis of worst case time complexity: The enqueue operation to insert incoming events in the sorted Q takes $log(p*a*\xi)$ time. Lines 5-10 maintain the context map by pair-wise comparison of the heads of $Q\,[i, j]$. The complexity of this operation is $(p*a)*(p*a\text{-}1)/2$. The predicate evaluation in line 11 has a complexity of $O(f(\Phi))$ where Φ is the predicate function. So, the worst case time complexity of *Algorithm1* is: $O((p*a*\xi)(log(p*a*\xi)+O(f(\Phi)+(p*a)*(p*a - 1)/2)))$.

C. Algorithm for Periodic $Conc_{pred}$ Detection

Algorithm2 is a centralized algorithm for periodically evaluating concurrent predicates considering asynchrony in event reporting. All events which occur during an epoch of period t (*i.e., for events with* $t_s<t$) are captured considering a maximum event reporting delay of Δ, and stored in the interval queue $Q\,[i, j]$. If an event, which arrives during (t+Δ), has $t_s>t$, then it is made to wait in the event queue, Q, before finally placing it in $Q\,[i, j]$, pending the predicate evaluation for the current period.

When an epoch ends at (t+Δ), the server temporarily closes the last queued intervals in Q [i, j] with the current time stamp, t, and then evaluates the attribute values of the intervals at the heads of all Q [i, j] to detect (i) whether any pair has matching attribute-values in which case a contextual link is added, and (ii) whether the predicate Φ is satisfied. After the first round of evaluation, some intervals are deleted from the heads of some of the Q [i, j] and another round of comparison is carried out among the updated heads of Q [i, j]. This process is repeated (line 9-24) until heads of all Q [i, j] are the latest intervals for the current epoch.

Algorithm2: Online Algorithm for Periodic *Conc$_{pred}$*

```
Event: ($P_i$, $A_j$, $Val$, $t_s$)
Interval: ($Val$, $t_s$, $t_f$)
Initialize:
        queue of events: $Q = <\perp>$
        queue of intervals: $(\forall p \ \forall a)\ Q[p, a] \leftarrow enqueue\ (default, t, \perp)$
Procedure ENQUEUE ($e$)
    (1) tail ($Q[p, a]$).$t_f \leftarrow t_s$
    (2) Enqueue ($Val$, $t_s$, $\perp$) in $Q[p, a]$
On receiving an event e from process $P_i$ at $P_0$ during epoch
    (3) ENQUEUE ($e$);
When epoch ends at t
    (4) wait Δ
    (5) for each e that arrived in [$t$, $t$+Δ] do
    (6)     if ($e.t_s < t$) then ENQUEUE ($e$)
    (7)     else ENQUEUE ($e$) after processing events in the current epoch
    (8) $(\forall p \ \forall a)$ tail ($Q[p, a]$).$t_f \leftarrow t_s$
    (9) repeat
    (10)    for all $a$ in $A$ do
    (11)        $\forall m,n \in P$
    (12)            if $(f(m.a = n.a))$
    (13)                add a link $L(A_j, Val, t_s, \perp)$ in the context map between $m$, $n$
    (14)            else
    (15)                close any existing link with current time stamp: $L(A_j, Val,$
    $t_s, t_f)$ between $m$, $n$
    (16)  if $\Phi((\forall p \ \forall a)$ head ($Q[p, a]$).$Val$) = TRUE then
    (17)      set ALARM ($\max_{p,a}$(head ($Q[p, a]$).$t_s$), $\min_{p,a}$(head ($Q[p, a]$).$t_f$))
    (18)  $t_x \leftarrow \min_{p,a}$ (head ($Q[p, a]$).$t_f$)
    (19)  if ($t_x < t$) then
    (20)      $(\forall p \ \forall a)$ | head ($Q[p, a]$).$t_f = t_x$, delete head ($Q[p, a]$)
    (21)      $stop \leftarrow$ FALSE
    (22)  else
    (23)      $stop \leftarrow$ TRUE
    (24) until $stop$
```

Deletion of intervals also helps to prevent overflow of a Q [i, j], which can hold up to ξ intervals per process per epoch. We observe that the interval which has finished first among the intervals at the heads of all Q [i, j] cannot overlap with any successor intervals and hence, no concurrency will be possible among them. So, we detect such interval(s) with earliest finish time at the heads of all Q [i, j], and delete it (or them) (line 18-23).

When a predicate is satisfied, our algorithm can detect the period of concurrency, i.e., the overlapping time of the intervals (T_{OLAP} in Fig. 4) over which Φ is defined. This is achieved by subtracting t_s of the interval with latest start time from the t_f of the interval with earliest finish time (line 17).

Analysis of worst case time complexity: The function ENQUEUE(e) which enqueues incoming events has a time complexity of $O(p*a)$. The *repeat* loop which spans lines 9-24 can be executed at most $O(p*a*(\xi-1))$ time. Lines 10-15 maintain the context map based on pair-wise comparison of the heads of Q [i, j]. The complexity of this operation is $(p*a)*(p*a - 1)/2$. The predicate evaluation in line 16 has a complexity of $O(f(\Phi))$ where Φ is the predicate function. Evaluating period of concurrency in line 17 has no extra cost. Each of the operations for detecting and removing time intervals (line 18-23) requires $O(p*a)$ time. So, worst case time complexity of the Algorithm2 is: $O((p*a*(\xi-1))(p*a + O(f(\Phi)+(p*a)*(p*a - 1)/2)))$.

D. General Notes on Conc$_{pred}$ Algorithms

With a maximum of ξ events per attribute per process, we are stepping through $O(p*a*\xi)$ states. To generate each state from the previous one in Algorithm2, it takes $p*a$ time. In Algorithm1, each state is created in $O(1)$ cost by merging the received event information with the state information of other attributes / processes.

To maintain a context map, we have to monitor multiple predicates. Algorithm1 can achieve that by simply repeating lines (5-12) – in a loop, iterating through the predicates.

Though we have assumed synchronized process clocks, in practice it is impossible to achieve complete clock synchronization and some skew always remains. Our algorithms can detect concurrent predicates considering skew between the clocks of different processes using the methods explained in [15]. However, in that case, we need to assume that the predicates hold unchanged for at least 2ε, where ε is the skew between different process clocks.

The authors in [15] have proposed two algorithms that can detect unstable predicates which retain their truth value for at least 2ε, where ε is the skew between different process clocks. The first algorithm aims to detect a global predicate at a predetermined clock value T. The second one is a centralized algorithm for global predicate detection where individual processes detect local predicates and then send the intervals, through which the predicate holds, to a central server which then detects global predicates.

V. Performance Evaluation

We have carried out extensive simulations to evaluate the performance of our proposed algorithms. We have considered creating context map based on the dynamic location context attribute as it quickly changes with time and the context map needs to be updated frequently.

A. Simulation Setup and Metrics

The network nodes are randomly scattered in a square territory. The total number of nodes is varied to examine the effect of system scale on the performance. For message routing, we have implemented a simple protocol based on the "least hops" policy, which is adopted in many classical routing protocols in ad hoc networks. A routing table is proactively maintained at each node.

TABLE II. SIMULATION PARAMETERS

Parameters	Values
Number of nodes, (N)	50, 100, 150, 200
Territory scale (m^2)	1500
Mean link delay (ms)	5
Max link delay (ms)	100
Transmission radius (m)	100
Routing Policy	Least hops
Mobility model	Random Waypoint
Node speed V (in m/s)	5, 10
Pause time (ms)	10, 50
Period of Predicate Evaluation (ms)	100

We assume that, every node has an id and a location attribute which is represented by its co-ordinates in the 2D simulation territory. Node 0 is the central server which keeps track of the location of other nodes and constructs the context graph. The territory is divided into 3x3 square grids which are considered as enclosed physical areas, like rooms. Nodes in the same grid are considered as co-located and they are linked with a co-location relation. When nodes moves across grids, the co-location relations change to trigger an event and the context map is updated accordingly. Our simulation parameters have been listed in Table II.

In this experiment, we measure the time delay in updating the context map using our two proposed algorithms with the help of the following metrics.

UD (Update Delay): It is the average time delay in milliseconds between the time a node changes location and the time the context map is updated.

We run each simulation for 20 simulation minutes and each point is obtained by averaging over 10 different runs. We do not consider node failure during the experiment.

B. Analysis of Performance Results

We plot the results of delay in updating context map for algorithms 1 and 2 for varied node speeds and pause times. Fig. 5 plots UD by varying N while keeping the node pause time as 10 ms.

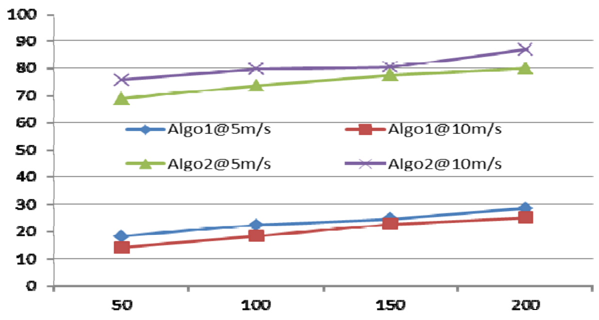

Figure 5. N vs. UD (Pause time = 10 ms)

We observe that for both the algorithms, UD increases with N and for the same N, UD increases with the node speed. UD increases with N as the central server needs to manage more events generated by higher number of nodes. For the same reason, at higher node speed, the nodes frequently move across grid boundaries and many new events are triggered. For the Algorithm 2, the observations are similar with a marked difference in the values of UD which is generally quite high. Since the update is periodic, the location changes occurring just after an update needs to wait till the completion of the period. This process increases the average updated delay of the context map.

Fig. 6 plots UD for varied N while the node pause time is 50 ms. The general trends of the graphs remain same in Fig. 5 and Fig. 6 However, the higher node pause time in Fig. 6 results in less UD for the same values of N and node speeds. This is because, the higher pause time means that the nodes move across grid boundaries less frequently resulting in lower number of event generations.

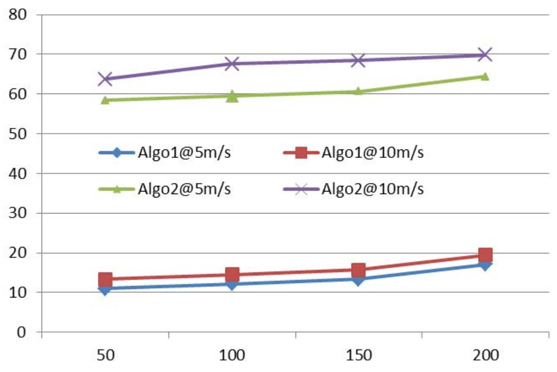

Figure 6. N vs. UD (Pause time = 50 ms)

In the next section, we have verified our simulation results by implementing the two algorithms on a wireless sensor networks testbed system.

VI. TESTBED SYSTEM FOR CONTEXT MAP

We have developed a ubiquitous searching and browsing framework (USBF) which uses context map for a demo application of a smart logistics network. In Logistics, it is always necessary to keep track of the goods being carried as well as the available infrastructure, like vehicles, personnel, etc. In this application, we consider that a logistic company (ABC Logistics Co.) divides its operating area (e.g., a city) into four delivery zones and the entire area is covered with a wireless sensor network. Sensor nodes can track the logistic vehicles collaboratively. We use our intelligent traffic system testbed (Fig. 7(a)) for this application.

In this demo, we use four smart cars (as logistic vehicles) embedded with sensor nodes, a smart mobile phone (Nokia XpressMusic 5800) with attached Nokia™ uSD card [22], and many sensor nodes attached to our intelligent traffic system testbed (Fig. 7(b)). Nokia uSD cards enable direct interfacing between mobile phones and sensor nodes. We assume that each car carries one driver, one laborer and a number of deliverable goods in separate packages. All the objects (cars, personnel and packages) are smart and have embedded sensing and communication capabilities. Each goods package has a RFID tag attached with it and contains sender's name and address, receiver's name and address, delivery due date, etc. Together the smart entities form a context map as shown in Fig. 7(c). While two cars of the company are linked as *co-owned*, the drivers, laborers, and packages in the car are linked with that car as *carried* entities. All the drivers and laborers are connected as *colleagues*. All these relationships are static as they do not change with time.

Each driver has a mobile phone in which they can see a map of the entire testbed. Each entity can be clicked and their attributes can be studied. The context map can be browsed by clicking an entity and following their contextual hyperlinks. When a car suffers a breakdown, the driver can use the context map to find out which other car of his company is in the same delivery zone concurrently with him. The concurrency relation can be specified using a conjunctive predicate. Assuming that driver of car1 (currently in zone1) wants to find whether car2 owned by his employer is also in zone1, by using the following conjunctive predicate:

(Car1.Owner = ABC Logistics & Car1.Loc = Zone1) & (Car2.Owner = ABC Logistics & Car2.Loc = Zone1).

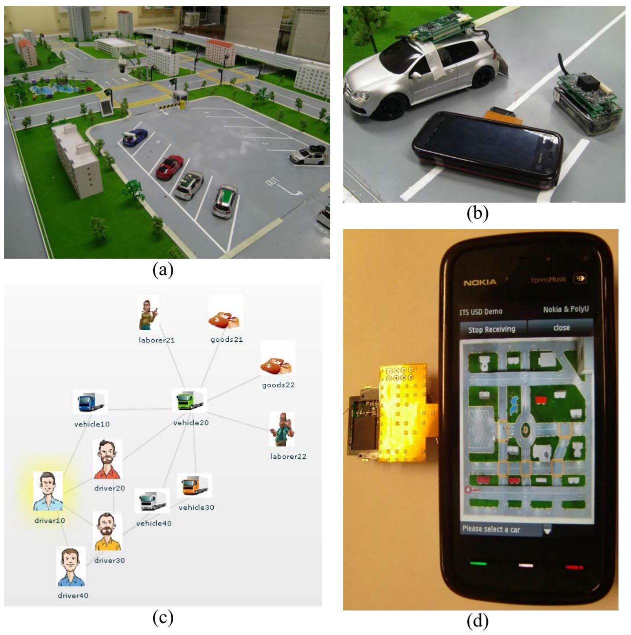

Figure 7. (a) Snapshot of our Smart Traffic System testebed, (b) Smart car, smart phone with uSD card, and sensor mote, (c) Context map for the logistic application, (d) Logistic vehicle tracking (using an area map of the testbed) through the mobile phone (vehicle shown with red circle and arrow)

If this predicate is satisfied, the context map will show a *co-location* link between car1 and car2 and this is a dynamic link. Similarly, other predicates can be specified comprising more than two cars and many other attributes. After the driver receives a list of logistic vehicles present in the same delivery zone, he can browse through them to get detailed information regarding the contents of the vehicle, delivery location, available space, driver contacts, etc. A video demo of the above application is available in [23].

In our testbed, the predicate detection and context map creation and maintenance is done in the following way. All the sensor nodes and the uSD card are pre-synchronized with negligible skew. The location attribute is updated by the smart car every time it changes, by sending a message to the central server. Whenever a new event (generated by location change of a car) arrives in the server, it runs Algorithm1 to detect location concurrency specified by the drivers. Cars which are concurrently in the same delivery zone are linked with a co-location relation in the context map. We have also tested for Algorithm2 by moving the cars continuously. The algorithm waits for a period of 5 seconds before evaluating the predicates and updating the context map. When using Algorithm2, the context map still contains old information even if many new events have occurred and refreshes the map after every 5 seconds. This observation is similar to the one obtained in simulation where the context map update delay is higher for Algorithm2. Algorithm1 also incurs some delay in updating the context map if the cars frequently move across zone boundaries, in which case the dynamic links are required to be updated. Otherwise, Algorithm1 performs well.

VII. 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 updating the contextual links in a timely and consistent manner is non-trivial in asynchronous pervasive computing environments. To resolve this problem, we proposed two centralized algorithms for online concurrency detection of contextual events. While one algorithm detects concurrent events as and when it happens, the other one periodically executes concurrency detection operation. Our proposed algorithms have low time complexity and can be applicable to a wide range of pervasive applications. We have evaluated our algorithms through extensive simulations and also through testbed experiments.

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.

Acknowledgement

This work is partially supported by RGC GRF Grant PolyU 5106/10E and China's Hi-Tech 973 project Grant 2009CB320702.

References

[1] C. Xu and S. C. Cheung, "Inconsistency detection and resolution for context-aware middleware support," *Proc. of ACM SIGSOFT International Symposium on Foundations of Software Engineering* (FSE), Lisbon, Portugal, Sep. 2005, pp. 336–345.

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

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

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

[5] Y. Huang, X. Ma, J. Cao, X. Tao, and J. Lu, "Concurrent Event Detection for Asynchronous consistency checking of pervasive context," *Proc. of IEEE International Conference on Pervasive Computing and Communications*, 2009.

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

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

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

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

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

[11] A.D. Kshemkalyani, "Immediate Detection of Predicates in Pervasive Environments, Journal of Parallel and Distributed Computing," (2011), doi:10.1016/j.jpdc/2011.09.004.

[12] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, vol. 21, no. 7, pp. 558-565, 1978.

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

[14] V. Raychoudhury, A. D. Kshemkalyani, and J. Cao, "Querying Context Maps using Relative Timing Predicates in Pervasive Environments," *Accepted for Publication in Proc. of 6th International Workshop on Middleware Tools, Services and Run-time Support for Networked Embedded Systems* (MidSens) to be held with Middleware 2011 Conference, December 12-16, 2011, Lisbon, Portugal.

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

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

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

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

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

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

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

[22] http://www.usdcard.org/

[23] http://imc.comp.polyu.edu.hk/pvc/doku.php?id=demonstration