Context Map for Navigating the Physical World

Vaskar Raychoudhury, Jiannong Cao,
Weiping Zhu, Ajay D. Kshemkalyani

Outline

- Context Map
- Our Research Focus
- Problem Assumption and Problem Definition
- Solution
- Simulations
- Conclusion

Smart Objects

- Smart objects are produced as a result of advances in
  - embedded sensing technologies,
  - wireless communications, and
  - mobile computing

- Smart objects are physical objects
  - capable of sensing, computing, and communication

Context Links

- Smart objects can be contextually inter-related
  - Contexts can be location, ownership, social connections, etc
  - Context can be static or dynamic depending on whether their value changes with time
  - e.g. if Mr. X and Mr. Y are present in the same location L, then there is a link between them with respect to the location context
  - If after some time, Mr. X and Mr. Y move away from each other, their contextual link may not exist any more
- Context links form a context map

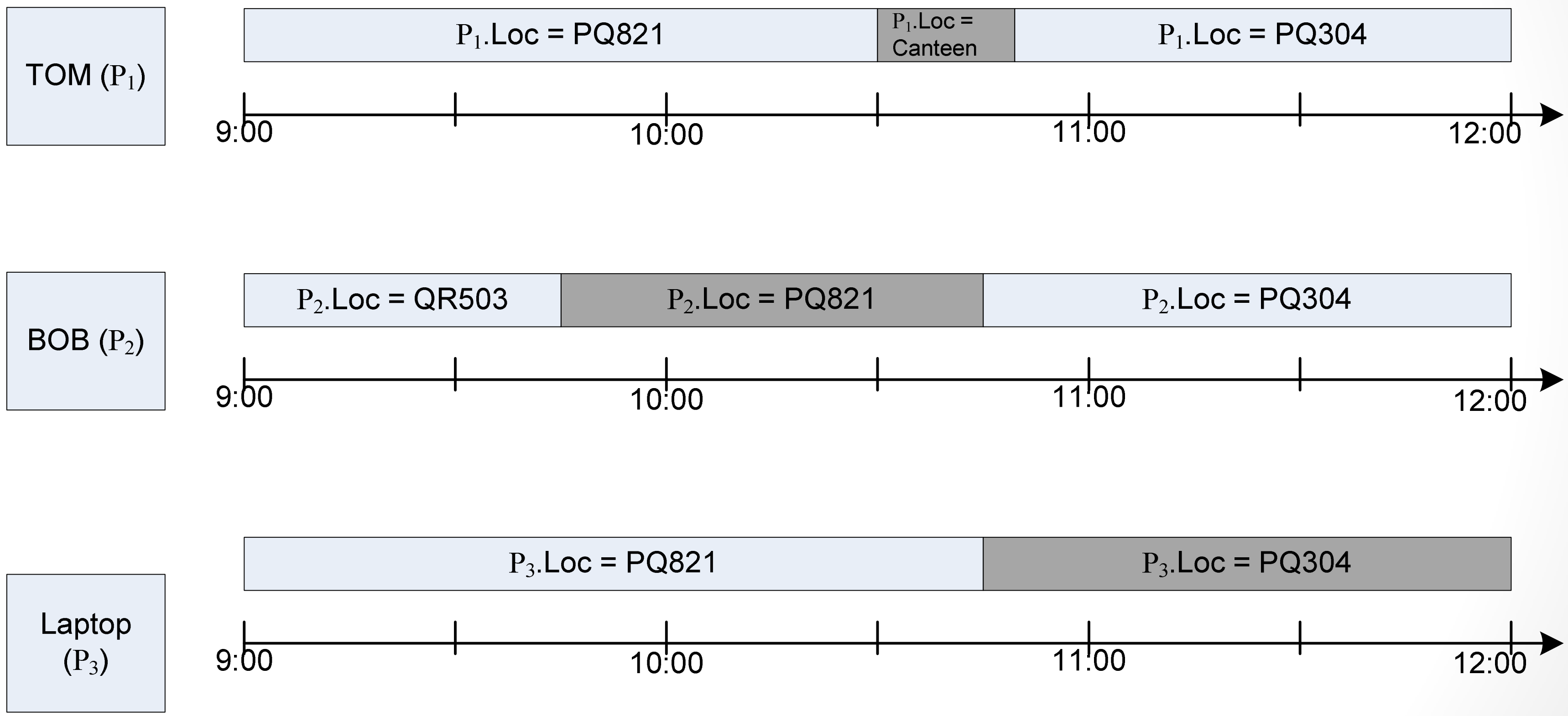
Context Map

- A context map represents a global snapshot of the physical world
- A global snapshot should contain one local state from each participating entity
- Using a common time axis, a global state can be specified
  - as a state occurring at the same time instant in each entity (or, concurrent), or
  - in terms of specific temporal relationships among the local states (or, relative)
- The global snapshot is also called an event in this paper.

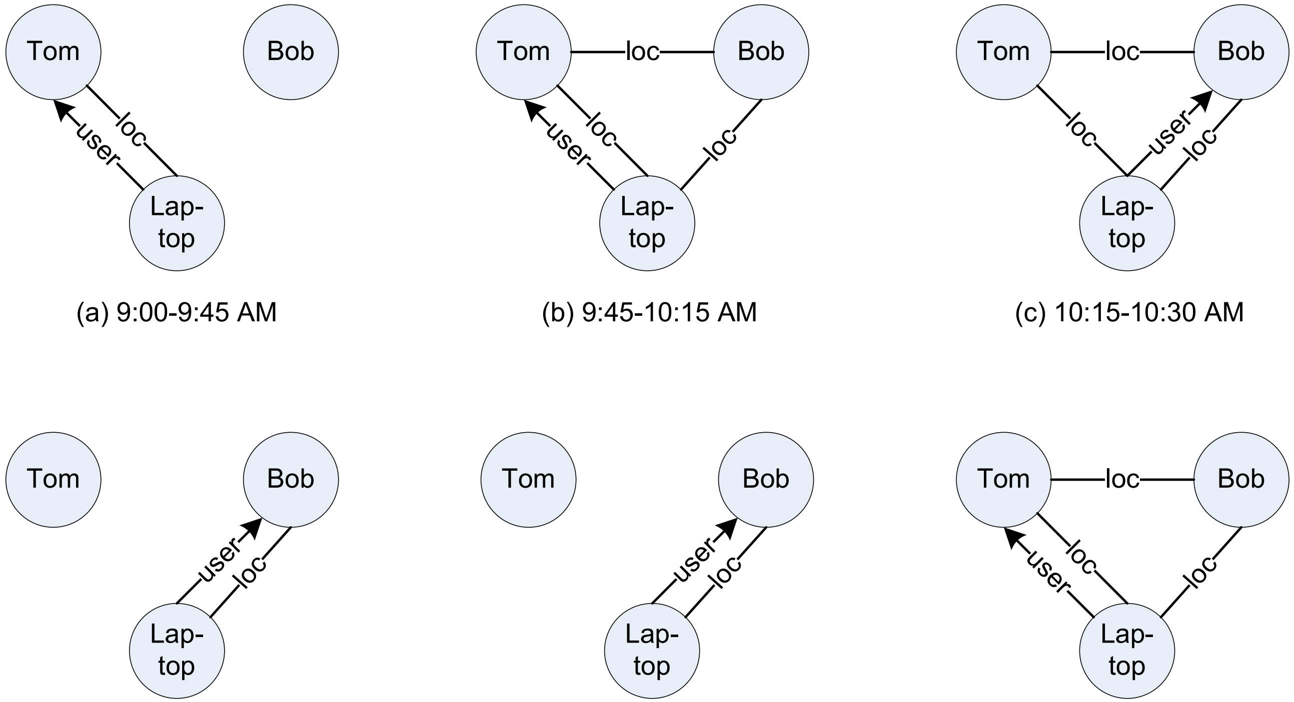
Example Scenario

- **Tom** enters his office **PQ821** at 9:00 am with a **laptop** borrowed from the office IT services for presenting at a meeting scheduled at 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 **PQ304** at 10:50 am.
- He finds that **Bob** has arrived there at 10:45 am and has set up the **laptop** for presentation.

Timeline for Example Scenario



Context Map for Example Scenario



(a) 9:00-9:45 AM

(b) 9:45-10:15 AM

(c) 10:15-10:30 AM

(d) 10:30-10:45 AM

(e) 10:45-10:50 AM

(f) 10:50 AM-12:00 Noon

Focus of Our Research

- In this paper, we have studied how to build the context map based on concurrent event detection
- We classified pervasive computing applications based on
  - event reporting delay, and
  - the event processing interval at the central server
- We proposed two online centralized algorithms for concurrent event detection in
  - An instantaneous manner (when the event happens), and
  - A periodic manner (detecting batch of events occurring in a time period)

Challenging Issues

- Consistent and timely maintenance of context map is non-trivial due to
  - Dynamic and asynchronous nature of pervasive environment
  - Incorrect detection of contextual events
  - Unreliability of wireless communication

Assumptions

- Various smart entities are connected wirelessly and asynchronous message passing
- Each entity has a set of static or dynamic context attributes
- Changes in those context attributes generate 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*
- Synchronized physical clocks are available among all smart objects

System Model

- Processes send event intervals to a central server, P0,
  - either periodically or
  - following a trigger-based approach
- A user queries for concurrent events which are specified through a predicate Φ, such that
  - Φ is explicitly defined on attribute value intervals that are implicitly related using concurrent timing relationships
- Event streams are “fused” at P0 and examined to detect Φ
- The context map is updated based on the truth value of Φ,

Types of Predicates

- Relational Predicates
  - can be true for any values of the context attributes, and cannot be evaluated locally
- Conjunctive Predicates
  - must be expressible in conjunctive form, i.e., as a conjunct over the local predicates Φ_i, where timing relations between intervals are included in the conjunction operation Λ^t
  - E.g. Conjunctive predicate: (Tom.*Loc* = PQ821 & Bob.*Loc* = PQ821 & Laptop.*Loc* = PQ821).
  - can be locally evaluated
- This work considers only conjunctive predicate based queries

Problem Definition

- **Problem *Conc*$_{pred}$**
  - Given a set of processes $P= \{P_1, P_2, \ldots, P_p\}$,, such that,
    - each process has a set of k attributes, $A = \{A_1, A_2, \ldots, A_a\}$,
    - 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 ($P_i.a_j$, $\forall P_i \in P$ $\forall a_j \in A$)
  - Identify a set of intervals I = $\{I_1; I_2; \ldots I_p\}$, where I_i is from process Pi,
    - such that there is some time instants within all these intervals at which Φ is true

Algorithms for $\text{Conc}_{\text{pred}}$

- We have proposed following two online algorithms
  - Algorithm for Trigger-based $\text{Conc}_{\text{pred}}$ Detection
  - Algorithm for Periodic $\text{Conc}_{\text{pred}}$ Detection
- Our algorithms consider asynchronous event reporting (bounded by Δ)

Data Structure for $Conc_{pred}$

- The central server maintains two different queues
  - One single *queue of events* (*Q*)
    - holds a list of incoming events sorted with respect to t_s
  - A number of *p*a* queues, called *interval queues* (*Q* [*i*, *j*])
    - captures the intervals generated by each attribute of each process
    - 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)

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
- The algorithm assumes that when a new event occurs the previous value of the context attribute, which was holding for a time interval, changes
- So, the previous time interval gets closed while a new interval starts which will continue until the next event trigger occurs
- Worst case time complexity (WCTC):
  - Enqueuing incoming events in the sorted Q => $log(p*a*\xi)$ time
  - Pair-wise comparison of the heads of $Q[i, j]$ => $(p*a)*(p*a-1)/2$ time
  - Predicate evaluation => $O(f(\Phi))$ where Φ is the predicate function
  - Total WCTC => $O((p*a*\xi)(log(p*a*\xi)+O(f(\Phi)+(p*a)*(p*a - 1)/2)))$

Algorithm for Trigger-based $Conc_{pred}$ Detection (Cont'd)

- The algorithm works in the following way
  - Incoming events at the server are enqueued in a ordered event *queue* (*Q)*
  - It then starts a timer of Δ to capture all delayed events which occurred within (t_s–Δ, t_s)
  - When the timer expires, the server transfers the event from the *Q* to *Q* [*i*, *j*], removing the previous head 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. 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
    - (i) whether any pair has matching attribute-values in which case a contextual link is added, and
    - (ii) whether the predicate Φ is satisfied

Algorithm for Periodic $Conc_{pred}$ Detection

- The algorithm periodically evaluates concurrent event 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 arrives during (t+Δ) and $t_s>t$, then it is made to wait in the event queue, Q, before finally placing it in Q [i, j]. The evaluation is pended for the current period.

Algorithm for Periodic $Conc_{pred}$ Detection (Cont'd)

- 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 until heads of all *Q* [*i*, *j*] are the latest intervals for the current epoch

Algorithm for Periodic $Conc_{pred}$ Detection (Cont'd)

- Analysis of worst case time complexity
  - The function ENQUEUE(*e*) which
  - Enqueuing incoming events => $O(p*a)$
  - The *repeat* loop => $O(p*a*(\xi-1))$
  - Pair-wise comparison of the heads of $Q[i, j]$ => $(p*a)*(p*a - 1)/2$
  - Predicate evaluation => $O(f(\Phi))$ where Φ is the predicate func.
  - Detecting and removing time intervals => $O(p*a)$
- Worst case time complexity of the algorithm
  - $O((p*a*(\xi-1))(p*a + O(f(\Phi)+(p*a)*(p*a - 1)/2)))$

Performance Evaluation

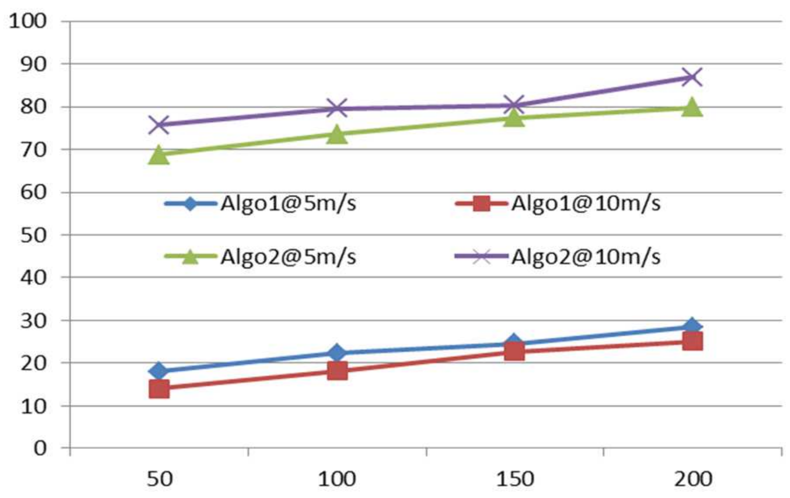
- Simulation setup
  - Every node has an id and a location attribute in the 2D simulation territory
  - Node 0 acts as the central server which keeps track of the location of other nodes and constructs the context graph
  - The territory is divided into 3x3 square grids which are considered as enclosed 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

Performance Evaluation

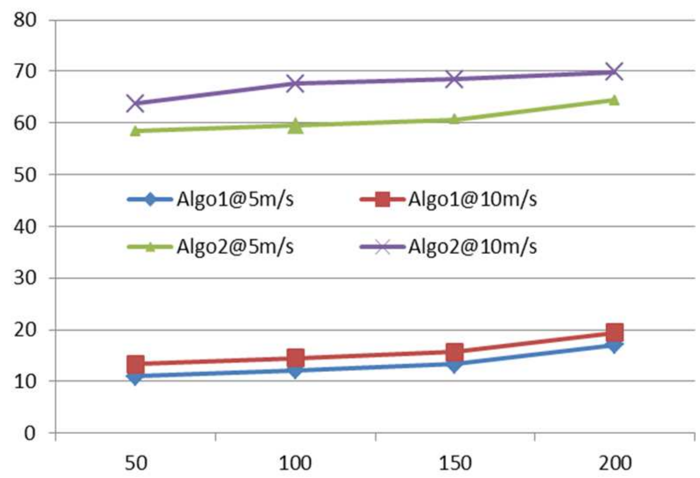
- Simulation parameters

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

Performance



N vs. UD (Pause time = 10 ms)
Higher node speed ensures higher UD



N vs. UD (Pause time = 50 ms)
Higher pause time ensures lower UD

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.

Conclusion

- In this paper
  - We introduce the context map to track the image of the physical world, so that queries can be run against the context map
  - We give instances of real-world problems in pervasive environments where complex timing relations are involved in queries on the context map
  - We show how to maintain the context map by simulation and testbed experiments

Thank You!

Email: vaskar@ieee.org

csweizhu@comp.polyu.edu.hk

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

Data Structure for $Conc_{pred}$

- 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*

Data Structure for $Conc_{pred}$

- 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 that of the other process through some user define function, *f*.

Concurrent and Relative Events

- Concurrent Event
  - concurrency event in terms of the location context of *Tom, Bob and the Laptop* : (Tom.*Loc* = Bob.*Loc* = Laptop.*Loc*)

- Relative Event
  - Consider a scenario where, after Tom enters his office, he has invited both Bob and Alex to take a look at his slides. After that they may go to the meeting together or separately.
  - So, the relative event in this example can be specified as - **Bob and Alex enter Tom's office AFTER Tom and they leave BEFORE Tom**