Asynchronous event detection for context inconsistency in pervasive computing

Daqiang Zhang

School of Computer Science,
Nanjing Normal University,
Nanjing 210046, China
E-mail: dqzhang@njnu.edu.cn

Zhangbing Zhou*

School of Information Engineering,
100083, China University of Geosciences (Beijing)
E-mail: zbzhou@cugb.edu.cn
*Corresponding author

Qin Zou

School of Computer,
Wuhan University, Wuhan 430079, China
E-mail: qinnzou@gmail.com

Tianyi Zhan

Department of Computer Science,
Nanjing University, Nanjing 210032, China
E-mail: zhanty@nju.edu.cn

Minho Jo*

College of Information and Communications,
Korea University, Seoul 136701, South Korea
E-mail: minhojo@gmail.com
*Corresponding author

Abstract: Event detection for context inconsistency is challenging in pervasive computing environments, where contexts are often noisy owing to fragile connectivity, node frequent movement and resource constraints. As a recent scheme, CEDA – Concurrent Event Detection for Asynchronous inconsistency checking (CEDA) – concurrently detects context inconsistency by exploring the *happened-before* relation among events. Nevertheless, CEDA suffers from several problems – unscalable from partial centralised detection manner, heavy computation complexity and false negative. To address these challenges, we propose in this paper the SECA scheme – asynchronous event detection for context inconsistency in pervasive computing. It puts forward a new type logical clock – snapshot timestamp – to check event relations, which enables it to be efficient in the scenarios where CEDA fails to. Meanwhile, SECA comes up with a lightweight update mechanism for the snapshot clock, which considerably reduces time and space complexity. Extensive experiments have been conducted and results show that SECA surmounts CEDA with respect to detection accuracy and scalability.

Keywords: context inconsistency; context-awareness; event detection; snapshot-based timestamp; ubiquitous computing.

Reference to this paper should be made as follows: Zhang, D., Zhou, Z., Zou, Q., Zhan, T. and Jo, M. (2012) 'Asynchronous event detection for context inconsistency in pervasive computing', *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 11, No. 4, pp.195–205.

Biographical notes: Daqiang Zhang is an associate professor at Nanjing Normal University, China. He received his PhD from Shanghai Jiao Tong University, China. His research interests include Internet of Things, cloud computing and ubiquitous computing.

Zhangbing Zhou is an associate professor at China University of Geosciences (Beijing). He received his PhD from Digital Enterprise Research Institute at Galway, Ireland. His research interests include process-aware information system, service-oriented computing, cloud computing and sensor network middleware.

Qin Zou received his PhD from Wuhan University, China in 2012. From Oct. 2010 to Oct. 2011, he was a visiting student at the Computer Vision Lab, University of South Carolina. His research interests involve computer vision, machine learning, ubiquitous computing and intelligent transportation systems.

Tianyi Zhan is a PhD student at Nanjing University. Her research interests include multi-agent system, mechanism design and algorithmic game theory.

Minho Jo is a Brain Korea Professor of College of Information and Communications, Korea Univ., Seoul. He received his PhD in Dept. of Industrial and Systems Engineering, Lehigh Univ., Pennsylvania, USA in 1994. He is an Editor of IEEE Network. He is the Founder and Editor-in-Chief of KSII Transactions on Internet and Information Systems. He is the Vice President of Computer Society of IEEK (Institute of Electronics Engineers of Korea).

1 Introduction

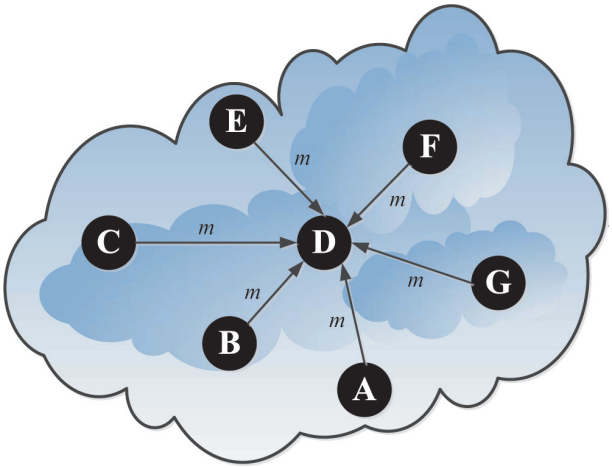
Growing convergence of wireless communication technologies, hand-held devices and embedded systems has fostered an attention to pervasive computing, which is a new computing paradigm shift from distributed system (Jian et al.,2012). The new paradigm aims at creating intelligent spaces so that users access services from spaces without awareness of underlying technologies. This kind of intelligence is chiefly achieved by context-awareness that refers to a mechanism that assists pervasive applications in adapting to varying contextual information. Contexts are pieces of information that captures the features of pervasive computing environments (Xu et al., 2008). Owing to the context noise caused by unreliable connectivity, resource constraints and dynamic context evolution, context inconsistency remains open (Zhang et al., 2010). For instance, *a RFID-based application may obtain two pieces of location contexts: the user is in the living room and the user is in the kitchen owing to the RFID data noise or signal interference* (Huang et al., 2009; Zhang et al., 2011; Zhang et al., 2012 and Jian et al., 2012).

A wide spectrum of schemes for context inconsistency checking have been proposed over the past three years. In Wang et al. (2004), Bu et al. (2004) and Bu et al. (2004), hidden rules and axioms from ontology have been exploited to check context inconsistency where contexts were denoted as assertions. In Xu et al. (2008), two policies *drop-all* and *drop-best* were proposed to resolve context inconsistency where contexts were modeled by tuples. Nevertheless, these two schemes did not delineate the context inconsistency checking. In Xu et al. (2006), a tree-based checking scheme on top of the first-order logic was reported, which detected context inconsistency by refining consistency trees using context constraints. To further extend the work, (Xu et al., 2010) reported the manners by which the context consistency was built and checked with partial constraints. However, most existing schemes are seriously limited by two problems. They require central control or centralised, which incurs their less scalability in large-scale pervasive computing environments with a huge number of nodes. Moreover, they implicitly assume that contexts being checked belong to the same snapshot so that they could not fairly measure the temporal relations among context events. *For the above RFID-based location acquisition, they could detect the context inconsistency with the assumption that these two pieces of location context took place at the same time.* This assumption is criticised in Huang et al. (2009) and Zhang et al. (2011) because it may not hold in some pervasive computing environments that concurrent events *(formal definition as Figure 5)* are normal owing to the asynchronous communication and network delay.

To eliminate the above two assumptions, CEDA (Concurrent Event Detection for Asynchronous inconsistency checking) (Huang et al., 2009) was proposed. Intuitively, it mapped context inconsistency checking into context event detection and checked concurrent context events based on the *happened-before* relation. CEDA has three limitations. Firstly, it checked event inconsistency in a centralised manner, incurring its less effectiveness in large-scale pervasive applications. Secondly, it introduced false negative because *happened-before* relation could hardly capture all event relations accurately. Finally, CEDA suffered heavy time and space complexity, resulting in its poor performance. Figure 1 illustrates the fairness of central-based systems in detecting context inconsistency among n nodes. All nodes are in the same environment and every node acquires one kind of context and delivers contextual information to central node D by m message. Thus, the node D will get $m * n$ communication overhead, n times as much as that of the normal node, (e.g., node A or F).

To address the challenges of the current schemes, we propose in this paper SECA scheme - asynchronous event detection for context inconsistency in pervasive computing environments, which is built based on time snapshots and logical clocks. SECA detects context inconsistency in a

Figure 1 The fairness of central-based systems in event detection for context inconsistency. The numbers in the figure refer to the corresponding link's bandwidth demand (see online version for colours)



distributed manner, which enables checking nodes not to be blocked or to become system bottlenecks. It adopts logical clocks rather than vector clocks to evaluate event relations. In order to be scalable, SECA customises logical clocks by holding the value part. Thus, it detects event relations that CEDA can and cannot support. Theoretical analysis and extensive experimental results show that SECA achieves higher detection accuracy than that of CEDA in a more scalable manner. To summarise, the main contributions of this paper are the following three-fold.

1 SECA scheme is significantly improved compared with the state-of-the-art detecting techniques, such as CEDA scheme from the performance aspect. Specifically, CEDA is a central-based checking system, which makes the payload of the system heavy. In contrast, SECA achieves its function in a fully distributed manner.

2 SECA scheme can detect false negative scenarios where CEDA fails to. We also conduct theoretical analysis to discuss about the reasons that SECA can while CEDA cannot.

3 SECA scheme respectively reduces CEDA's complexity of time and space from O (n^2) to O(n) and from O(1) to O(n), where n is the number of nodes in a pervasive network.

Preliminary result of this paper has been reported in our previous work (Zhang et al., 2012 and Zhang et al., 2011). This paper gives a more comprehensive presentation and discussion about the asynchronous event detection for checking the context inconsistency, including details of definitions, tractable algorithms and evaluation of the algorithms.

The remaining of this paper is structured as follows. Section 2 provides an overview of the existing work. Section 3 briefly introduces the background knowledge that is the basis of our work. Section 4 presents the design of SECA scheme, following by theoretical analysis. Section 5 reports our extensive experimental results. Section 6 concludes the paper and point out directions of the future improvement.

2 Related work

Context-awareness is a main mechanism for pervasive computing that enables people to unobtrusively access services without the awareness of underlying technologies. Noisy contexts usually lead to context-aware applications incongruous behaviours and user perplexed feelings. Hitherto, context inconsistency detection has drawn increasing attention in recent years.

A bunch of schemes have been proposed in the existing literature. In Wang et al. (2004) and Gu et al. (2004), ontology was used to model contexts and their properties. In Bu et al. (2006) and Bu et al. (2006), context consistency was specified by ontology assertions such that it could be checked by hidden rules and axioms in ontology. In Xu et al. (2006), context consistency was modeled by tuples and inconsistency checking was mapped to the comparison among the elements in tuples. This kind of context inconsistency was resolved in Xu et al. (2008), where three resolution policies including *drop-all* and *drop-best* had been proposed. To further detect context inconsistency based on first order logic, (Xu et al., 2010) reported the work that built context consistency trees and refined these trees by checking partial constraints. However, most existing schemes suffer from a couple of problems. They are heavily centralised or require central control, which incurs their ineffectiveness in large-scale pervasive environments or pervasive applications. Moreover, they do not consider the event relations, particularly temporal relations. As a matter of fact, most of them implicitly assume that contexts being examined belong to the same snapshot. This assumption may not always be satisfied in pervasive computing environments, which are characterised by asynchronous cooperation and schedule. Note that DCCI (Zhang et al., 2011) reported the work that detected context inconsistency in a peer-to-peer manner. However, this work also relies on the above two assumptions.

CEDA (Huang et al., 2009), as a derivation of Huang et al. (2011), took event temporal relations into account and presented a scheme on the basis of *happened-before* relation. It consisted of a checker process that determined context inconsistency and several normal processes that reported the values of event vector clocks. However, CEDA was a semi-centralised scheme in which the checker process might be the system bottleneck when the system scale increases. Another prominent issue is that its communication overhead was heavy, since normal processes reported their observations from time to time owing to the frequent happened context events. Taking RFID for example, a reader gets around 50 values each second from the same tag as usual (Zhang et al., 2011). In addition, CEDA ignored some false negative scenarios, which affected its detection accuracy.

It is worthwhile to mention that our work SECA is similar with IEEE 1516 time management (Fujimoto, 1998 and Group, 2000) with several differences.

- Firstly, we share the similar idea in time management in IEEE 1516. We focus on the concept of snapshot timestamp, as well as its characteristics with theoretical analysis. While IEEE 1516 emphasises on the usage of time management with distributed time clocks.
- Secondly, we develop and implement the snapshot clock for concurrent event detection, while IEEE 1516 is a standard that does not provide a way to checking event concurrency detection. Actually, IEEE 1516 is designed for simulations to ensure that temporal aspects of the system under investigation are correctly reproduced by the simulation model.
- Finally, SECA is an independent scheme and can be customised for personal computers and smart devices. While IEEE 1516 relies on federates and the Runtime Infrastructure (RTI), because it is just a part of IEEE 1516 standard. IEEE 1516 time management can be extracted as an independent module after modification. In addition, IEEE 1516 is more complex than SECA, providing various functions that are not mentioned in SECA, e.g., clock synchronisation and time compensation.

Consequently, we find it appropriate to revisit the event relation in asynchronous pervasive computing environment. Owing to the absence of a global clock and absence of 100% accurately synchronised clocks, logical clock was introduced in Lamport (1978) and *happened-before* was designed to measure event relations. In this paper, SECA customises a logical clock for event detection of context inconsistency. Theoretical analysis and experimental results show that SECA address the issue of context inconsistency detection with an accurate and scalable manner.

3 Background

In this section we introduce *happened-before* relation. Then we give an overview about the logical clock and Lamport's algorithm. Finally, we briefly present what vector clock is.

3.1 Happened-before relation

The *happened-before* relation is coined in Lamport (1978). It refers to a way of ordering events by the potential causal relation of pairs of events in a concurrent system, particularly asynchronous distributed systems. Suppose a single process is a set of events with a priori total ordering, the events of a process form a sequence and sending or receiving messages in a process is an event (Zhou et al., 2008). Thus, the *happened-before* relation denoted by is given as Definition 3.1.

Definition 3.1 *The '→' among events of a system is a relation such that:*

If b and c are events in the same process and b occurs before c, then $b \rightarrow c$.

If b is sending a message by one process and c is the receipt of the same message by another process, then $b \rightarrow c$.

If $b \rightarrow c$ *and* $c \rightarrow d$, *then* $b \rightarrow d$.

Note that any event cannot occur before itself, i.e., $b \nrightarrow b$ for any event b. Two events *b* and *c* are *concurrent* if both $b \nrightarrow c$ and $c \nrightarrow b$. In fact, the *happened-before* relation is a kind of strict partial order, which is inherently *transitive, irreflexive* and *asymmetric.* Note that the processes that compose a distributed system have no knowledge of the *happened-before* relation unless they are ordered by a logical clock, e.g., Lamport clock or vector clock (Fidge, 1988 and Mattern, 1994). Even though one event *b* occurs physically earlier than another event *c,* this does not imply that $b \rightarrow c$. For the same reason, $b \rightarrow c$ neither indicates that the event *b* occurs physically earlier than the event *c.*

3.2 Logical clock

Logical clock is introduced in distributed computing owing to the lack of global physical clocks and completely accurate synchronisation clocks. It is a mechanism for capturing causal and chronological relations between events. It is defined as follow:

Definition 3.2 *Given an event a in a process* P_i, *the corresponding logical clock* C_i *is a function that assigns a number* $C_i(a)$, *where the number is regarded as the time when the event occurred.*

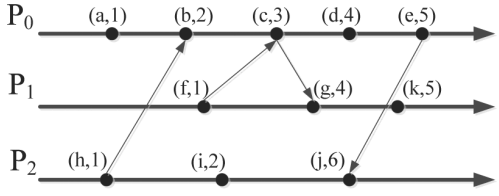
By Definition 3.2, the clocks of the entire system is represented by the function *C* that assigns to any event *a* the number *C(a)*, where $C(a) = C_{j(a)}$ if a is an event in the process P_j. Thus, only when one of following conditions is satisfied, the *happened-before* relation is true:

1 Events *a* and *b* are in the same P_i and *a* occurs before *b*, then $C_i(a) < C_i(b)$.

2 Event a in the process P_i sends the message to an event *b* in the process $p_{j,\text{ then}}\ C_i(a) < C_i(b)$.

Let $P_{0,\ P1}$ and P_2 be three processes and *a, b, ...,k* be events. All events are counted by Lamport's logical clock algorithm. Figure 2 illustrates an example using Lamport's logical clock, which partially orders the events happened in P_0, P_1 and P_2. Lamport's logical clock algorithm gets local events sequenced, e.g., *a* and *b* events. However, it causes an issue of identical timestamps and thus it hardly states which events are causally related. For instance, whether events *a* and f may have the same timestamps or not. To address the identical timestamp issue, vector clock is introduced.

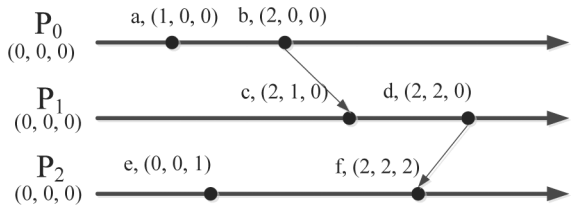
Figure 2 An example of Lamport's logical clock



3.3 Vector clock

Vector clock refers to a mechanism for partially ordering events in a distributed system and detecting causality violations. The vector clock of a holistic system is an array/ vector of N logical clocks and every process maintains a clock. A local smallest possible value copy of the global clock-array is kept in every process. Figure 3 shows an example of vector clock.

Figure 3 An example of vector clock



$$VC(b) < VC(c) \Leftrightarrow \forall j[VC_j(b) \le VC_j(c)] \wedge \exists k[VC_k(b) < VC_k(c)] \quad (1)$$

Let *VC* be a function of assigning the number *VC(a)* to an event a, where $VC(a) = VC_j(a)$ if a is an event in the process P_j. Equation 1 gives the judgement of vector clocks of two events b and c. Thus, *happened-before* relation in vector clock is given by Theorem 1.

Theorem 1 *If VC(b) < VC(c), then* b → c.

Vector clock is promising for event ordering, but it has several conspicuous issues. First, it requires that every process maintains an array to record the logical clock values of all processes. Second, it is designed under an assumption of a fixed set of participants (Almeida et al., 2008). This assumption may not be held in asynchronous pervasive computing environments, where nodes are able to join or leave pervasive networks randomly. The same issue appears in Lamport's logical clock. Consequently, these logical clock algorithms are inappropriate to be used in pervasive scenarios due to changing numbers of participants and churns.

4 Asynchronous event detection for context inconsistency in pervasive computing

In this section, we propose a scheme for event detection for context inconsistency in pervasive computing environments. We introduce the system model and then depict the system design in detail, followed by discussions.

4.1 System model

Pervasive computing environments are modeled as a loosely-coupled distributed system, where physical entities (e.g., objects and users) sense environments and pervasive infrastructures handle sensor readings and deliver services to pervasive applications. A pervasive computing environment is composed of a set of n asynchronous processes P_1, P_2, ..., P_n, which communicate with each other through message-passing (Zhou et al., 2012). The processes do not share a global memory and communicate solely by exchanging messages. The communication delay is finite but unbounded. An event e in the process P_j is modeled by intervals using boolean predicate E_j. The event e is occurring in the process P_j when E_j is true. Otherwise, the event does not exist. The events are modeled by means of intervals. The beginning and the end of an event are denoted as *lo* and *hi*, respectively.

4.2 Modeling events by intervals

The event is detected by a boolean predicate B_i. The event is happening in the process P_i when B_i is true and it does not occur when B_i is false. The event is represented by an interval, denoted by *lo* and *hi*, corresponding to the beginning and the end of the interval. The notations in the design of the proposed scheme are given in Table 1.

Table 1 Notations in the design of SECA algorithm

Notation	*Explanation*
n	the number of processes
P_i	the ith process involved in the context inconsistency detection
lo	the beginning of an interval
hi	the end of an interval
I	the interval that denotes the time period between two successive events
E_i	event on Pi
$SC_i[1...n]$	snapshot clock on P_i

4.3 System design

This subsection presents snapshot timestamp and its update policy and then introduces snapshot-based event detection.

Snapshot Timestamp is an implementation of logical clocks, where all nodes maintain a logical clock. In the system of snapshot clocks, the time domain is denoted as a set of *n*—dimensional, non-negative integer clock. Each process P_i maintains a snapshot clock $S_i[1..n]$, where $S_i[k]$ is the *k*th local logical timestamp and describes the logical time process at the process P_i. The process P_i updates its snapshot clock by the following rules:

- Before sending a message, the process P_i updates its local clock by

$$S_i[k] = S_i[k-1] + d(d > 0), \quad (2)$$

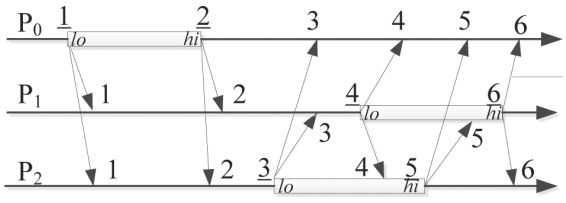
  where the default value of d is 1. Then, P_i piggybacks a message m with its snapshot clock to the remaining nodes in the same environment.

- When receiving a message (m, $S_j[send]$) from the process P_j, the process P_i gets the snapshot timestamp at the receiver point as:

$$S_i[receiver] = max(S_i[k], S_j[send]), (1 < k < n) \quad (3)$$

Figure 4 illustrates the update policies of snapshot clock algorithm, where events are represented by the starting and the end of intervals, i.e., *lo* and *hi*. When the process P_0 would like to send a message, it will automatically increment the value of its snapshot clock and then forwards it to the processes P_1 and P_2.

Figure 4 An example of snapshot-clock update mechanism



4.3.1 Basic properties

Isomorphism. Evidently, by comparing timestamps (i.e., an array of *n*– elements), the snapshot clock keeps its property of isomorphism. The relations between timestamp intervals include two ordering relations represented as '≤' and '<' and one concurrent relation denoted as '||'.

Property 4.1 *Given two snapshot timestamp Sp and Sq, the isomorphism of the snapshot clock is given as:*

$$Sp \le Sq \Leftrightarrow \forall_i Sp[i] \le Sq[i]$$

$$Sp < Sq \Leftrightarrow Sp \le Sq \text{ and } \exists_{i,i'} Sp[i] < Sq[i']$$

$$Sp \parallel Sq \Leftrightarrow \text{ not } (Sp < Sq) \text{ and not } (Sq < Sq)$$

Happened-before relation. Recall that relation → partially orders the set of events in a distributed execution. Snapshot timestamp based events in a distributed system satisfies Theorem 2.

Theorem 2 *Suppose two events b and c have timestamps Sp and Sq respectively, then:*

$$b \rightarrow c \Leftrightarrow Sp < Sq$$

$$b \parallel c \Leftrightarrow Sp \parallel Sq$$

Proof: According to the update policies of snapshot clocks, the *happened – before* relation holds. □

Consequently, an isomorphism property exists between the set of partially ordered events produced by a distributed computation and their timestamps. This is a powerful and interesting property of snapshot clocks. By checking timestamps, we are able to get the event concurrent relations. For instance, let the events *b* and *c* being occurred at the processes P_i and Q_j, respectively. They are assigned of timestamps S_p and S_q. Then, the *happened – before* relation between these two events is satisfied in Theorem 3.

Theorem 3

$$b \rightarrow c \Leftrightarrow Sp[i] < Sq[i]$$

Proof: Straightforwardness. □

Note that the concurrent relation between the events *b* and *c* can be detected by comparing their timestamps (e.g., *Sp* and *Sq)*. However, this is an inappropriate scheme owing to two reasons. One is that it is complex to compare snapshot timestamps. The other is that it may incur false negative when using the *happened – before* relation, which will be discussed in the discussion part. In order to easily detect concurrent events, we propose an event concurrence detection mechanism as presented in Theorem 4.

Theorem 4 *Given two events b and c in the processes P_i and P_j and these two events communicate each other.*

Assume the event b sends a message to the event c with its timestamp $S_{pb}[i]$. Then:

$$b \parallel c \Leftrightarrow (c_{lo} \le Sp_b[i] < c_{hi}$$

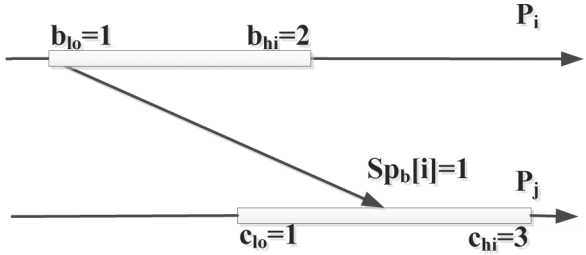
Proof: There is a message from the event *b* to the event c. According to the update policy of snapshot clocks, $Sp_b[i]$ is the maximal event timestamp between the timestamps of the events *b* and c, respectively. Thus, the value of $Sp_b[i]$ is not less than c_{lo}. Because the message is handled by the event c, the value of *Spb[i]* must be less than c_{hi} □

4.4 Snapshot-based Concurrent Event Detection

In this section we present the SECA scheme asynchronous event detection for context inconsistency in pervasive computing environments. SECA is built upon snapshot timestamps and enables all nodes to detect concurrent context inconsistency events without central control or centralised hierarchy. Thus, SECA dramatically reduces communication complexity and successfully avoids the risk that the central nodes are easily to be the bottleneck of a pervasive computing system.

The distributed architecture of SECA suggests that every process will automatically check context inconsistency. The basis of context inconsistency detection lies in the fact that: when two events *b* and c are concurrent, *b* and c can satisfy Theorem 4. Figure 5 illustrates the fact that the events *b* and c are concurrent.

Figure 5 Concurrent events *b* and *c*, where $I_c.lo \le I_b.x < I_c.hi$.



The pseudo-code of SECA scheme is given in Algorithm 1, which includes three parts: event processing (lines 2–8),

message processing (lines 9–19) and context inconsistency checking (lines 20–26).

The event processing refers to a process that updates its snapshot clock when an event occurs within its life span. To be specific, the process updates its snapshot clock, the event queue *EQ,* as well as interval queue *IQ*. by broadcast (e.g., SECA offers a *System-Broadcast* primitive). Note that we do not provide a function to reduce the message complexity in SECA scheme owing to its scale is not large. For large-scale applications of context inconsistency detections, to further reduce the message complexity, we may design a preliminary procedure to let every process realise which processes it should communicate with for inconsistency detection.

There are two kinds of message exchange actions: sending and receiving. The sender is in charge of updating the event queue and interval queue (see steps 11–14). Correspondingly the receiving process modifies its snapshot clock by picking the maximal timestamp value between the snapshot timestamps of the sender and receiver processes (see steps 15–19). Note that the actions of senders and receivers are incorporated together in the pseudo-code.

The third part of Algorithm 1 corresponds to the context consistency detection. Since the elements in *EE* implicitly satisfy Theorem 4, we output the event pairs simply by a validation check.

4.5 Discussions

Thus far, we have presented the design of SECA scheme in previous sections. Does SECA solve false negative caused in CEDA scheme? Does SECA detect context consistency accurately in pervasive computing environments? We investigate these issues with theoretical study in this section. We further evaluate SECA scheme by extensive experiments in the following Section 5.

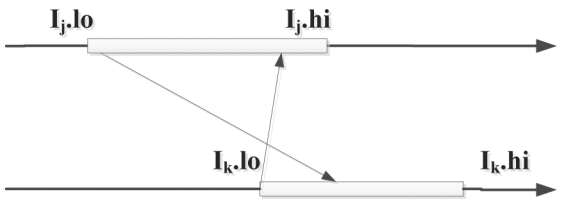
4.5.1 False negative in Happened – Before-based context consistency detection

Given n intervals $I_1, I_2, \ldots, I_n$, CEDA checks concurrent context consistency events by Equation 4, which is defined upon the *happened – before* relation.

$$(I_j.lo \rightarrow I_k.hi) \wedge (I_k.lo \rightarrow I_j.hi), \forall 1 \leq j \neq k < n. \quad (4)$$

The case of interval overlaps, which is characterised by concurrent events, is illustrated by Figure 6.

Figure 6 Overlapping intervals that can be detected based on *Happened – before* relation in CEDA scheme



Algorithm 1: SECA checks context inconsistency in normal processes

Input: $P = \{P_1, \ldots, P_i, \ldots, P_n\}$, a set of processes in a pervasive system
n, the number of processes in the system;
$EQ[]$, an event queue;
$IQ[]$, an interval queue;
$EE[]$, pairs of events which have communication with each other;
$S[]$, a list of snapshot timestamps;
Output: A set of concurrent events
$C = \{< e_x, e_y >\}$

```
1  begin
2    /* When an event e occurs at P_i                          */
3    if P_i <-occur- e then
4      /* suppose it occurs at timestamp k with
           id e^i_id                                            */
5      S_i[k]=S_i[k-1]+1;
6      EQ[i].push(e^i_id, S_i[k]);
7      IQ[i].push(e^i_id, lo=S_i[k], hi=S_i[k]+1);
8      System_Broadcast(P_i, S_i);
9    /* Upon the process P_i receives a message
        from the process P_j                                    */
10   if P_i <-msg- P_j then
11     (e^j_id, S_j[k]) = EQ[j].getTop();
12     S_j[k+1] = S_j[k]+1;
13     EQ[j].push(e^j_id, S_j[k+1]);
14     IQ[j].current = (e^j_id, lo, max(hi,
       S_j[k+1]+1);
15     S_i[receive] = max(S_i, S_j[k+1]);
16     if e^j_id is received by e^i_id then
17       (e^i_id, lo, hi) = IQ[j].pop();
18       IQ[i].push(e^i_id, lo, max(hi, S_j[k+1]);
19       EE.push(e^i_id, e^j_id);
20   /* Context inconsistency detection                        */
21   while (!EE.IsNullOrEmpty()) do
22     < e^i_id, e^j_id > = EE.pop();
23     if (IsValid(e^i_id)) && (IsValid(e^j_id)) then
24       C.push(e^i_id, e^j_id);
25     /* Output concurrent events                             */
26     Unique(C);
```

In some cases, intervals are overlapped and events are concurrent, but CEDA cannot detect them. This is notorious for *false negative* phenomena, thus definitely brings down the detection accuracy of the CEDA scheme. This is because Equation 4 cannot detect these overlapping intervals, although they are mutually across.

Figure 7 illustrates three kinds of false negative scenarios, where CEDA scheme fails to check context consistency correctly In Figures 7(a), 7(c) and 7(c), two events in two processes satisfy

$$(I_j.lo \rightarrow I_k.hi) \wedge (I_k.lo \nrightarrow I_j.hi), (I_j.lo \nrightarrow I_k.hi) \Lambda$$
$$(I_k.lo \rightarrow I_j.hi), \text{ and } (I_j.lo \nrightarrow I_k.hi) \Lambda (I_k.lo \nrightarrow I_j.hi),$$

respectively. These two event pairs occur concurrently but Equation 4 fails to detect them. On the contrary SECA scheme is capable of finding these concurrent context events successfully As for Figures 7(a) and 7(b), SECA compares the message timestamps of senders with the lo and hi of the receivers and then find the concurrency. Note that concurrency in Figure 7(c) is challenging to detect. This kind of concurrency is mainly caused by the message delay.

As a matter of fact, there are 25 temporal interaction of intervals in a distributed system (Kshemkalyani, 1996; Zhang et al., 2012 and Chen et al., 2012), as shown in Figure 8. We have checked these 25 temporal interaction of intervals and find that 16 temporal interaction of intervals can satisfy the requirement shown in Equation 4. This implies that these 16 temporal interaction of intervals can be accurately recognised by the happened – before relation. However, the rest of 9 temporal interaction of intervals, i.e., IA, IB, IC, ID, IE, IF, IJ, IH and IK labeled in Figure 8, may be overlapped in physical time, but the happened – before relation falls short of checking them. For those 16 temporal interaction of intervals within concurrent events, SECA scheme can detect them using the snapshot timestamp and hence may be regarded as a general solution. For the rest temporal interaction of intervals, it remains open to research community.

4.5.2 Complexity

Taking a panoramic view of the SECA scheme, it does not rely on central control to check context consistency.

All processes involved in a pervasive system are equal and check context consistency by snapshot clocks. Every process requires $O(1)$ space complexity to maintain snapshot timestamps and $O(n)$ time complexity for every context consistency event detection.

To further evaluate the time and space complexity of the proposed scheme, we have implemented the detection schemes by means of physical clocks, vector clocks and snapshot clocks, labeled as PCA, CEDA and SCA, respectively. Table 2 compares the PCA, CEDA and SECA in terms of their *clock synchronisation, handling the occurrence of an event, detecting overlapped intervals and concurrent events a*nd *false negative.* By comparison, SCA significantly reduces the time and space complexity concerning event processing and context consistency checking. Meanwhile, SECA cuts off a half of the possibility of false negative generated in CEDA scheme.

Table 2 Comparison of PCA, CEDA and SECA with respect to checking context consistency events

	PCA	*CEDA*	*SECA*
Synchronisation	✓	×	×
An event occurs	×	O (n)	O(1)
Concurrent event	O(n)	O (n^2)	O(n)
False negative	×	$\|overlap\| < \varepsilon$	$-\varepsilon < overlap < 0$

Figure 7 Three kinds of false negative scenarios caused by *happened – before* relation in CEDA scheme. Concurrent events in these scenarios can be accurately detected by SECA scheme (a) False negative scenario one between two concurrent events; (b) False negative scenario two between two concurrent events and (c) False negative scenario three between two concurrent events

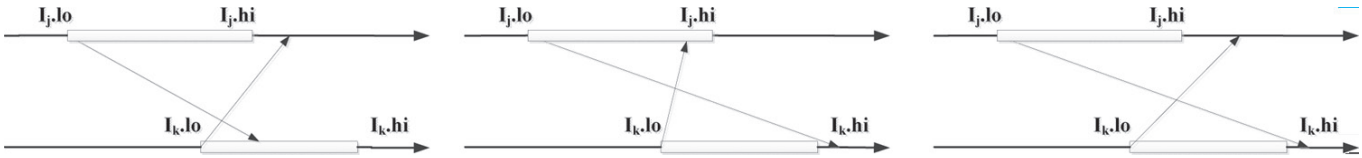
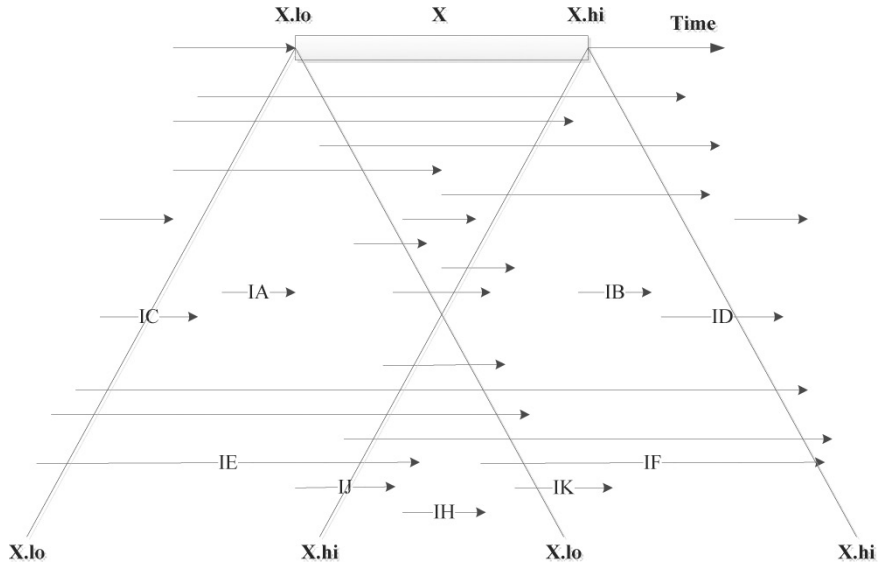


Figure 8 Temporal interaction of intervals in a distributed system



5 Experiments

We conduct extensive experiments in this section to further evaluate whether SECA is appropriate to context-aware applications in asynchronous pervasive computing environments. Particularly, this section will evaluate:

- to what extent the detection accuracy is that SECA scheme can achieve
- whether SECA outperforms CEDA in terms of detection accuracy and computation.

5.1 Experiment setup

A smart building scenario is simulated where users move around randomly. The duration of users' staying in an office follows the exponential distribution. In view of that user location is regarded as the most important type of context in asynchronous pervasive computing environments (Xu et al., 2009; Ni et al., 2004; Want et al., 1992 and Ji ,2011), user location is our focus. The environment is equipped with RFID devices and every user carries a RFID tag such that the location context is collected timely and correctly. The RFID data concerning user location is generated with controlled error rates of 10%, 20%, 30%, 40% and 50% by using the mechanisms presented in the existing literature (Xu et al., 2010 and Rao et al., 2006). A constriction is implanted that *a user cannot have two difference locations at the same time.* Table 3 reports the experimental settings in detail. Note that some parameters are not listed in Table 3, *e.g., lo and hi,* since they are included in the design of SECA scheme, illustrated in Algorithm 1. Meanwhile, the detection accuracy is evaluated as the average value for all processes in all network nodes.

Table 3 Experimental settings

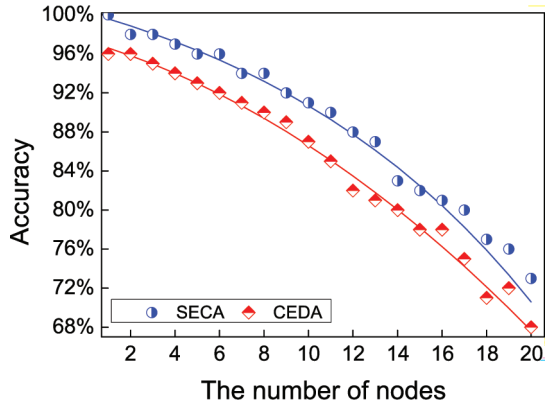
	Experimental settings
The number of processes	2–15
The event duration	5 – 120s
The message delay	0.01 – 655.36s
Checking Devices	Windows 7 Ultimate Intel Core2 CPU, 1.67 GHz Memory: 2 GB Disk: 512 GB

5.2 Overall performance

A series of experiments are designed to check the detection accuracy of SECA and whether it performs better than CEDA. Given that the experiments shed light on detecting concurrent events of user locations, we limit the number of nodes attending for the same contexts from 2 to 20. Every node runs two detection process instances. Every event has a random life span from 20 to 120 seconds and every message suffers a random delay between 0.25 to 8 seconds. The following experiments employed the same setting without explicit declaration.

Figure 9 illustrates the performance results with tuning the number of nodes from 2 to 20. Both CEDA and SECA schemes achieve high level of detection accuracy, showing a slightly downward trend. This indicates that vector clocks and *happened – before* relations are efficient for detection of concurrent context inconsistency events. Meanwhile, SECA gets higher level of accuracy than CEDA scheme, which is attributed to the exclusion cases that CEDA scheme fails to detect are checked by snapshot clocks in SECA scheme.

Figure 9 Overall performance of SECA scheme by increasing node participants

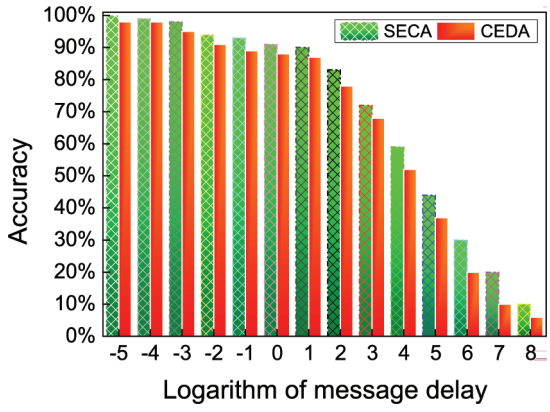


5.3 Detection performance with varying message delays

In this section, several experiments are conducted to investigate how the changes of the average message delay affect the concurrent event detection of the proposed scheme.

As shown in Figure 10, both SECA and CEDA schemes reduce their detection accuracy with the increase of message delay (note that the x–axis is the logarithm of the message delay, counted by seconds). In all experiments, SECA achieves higher level of accuracy than CEDA, owing to its snapshot-based timestamp checking mechanism. As the logarithm of the message delay is between –2 to 3, SECA gets a better detection accuracy with less communication overheads. Meanwhile, in view of the pervasive network scale, we hereby set the value of the logarithm of message delay as 0.25 to 8 seconds.

Figure 10 Detection accuracy with varying message delays

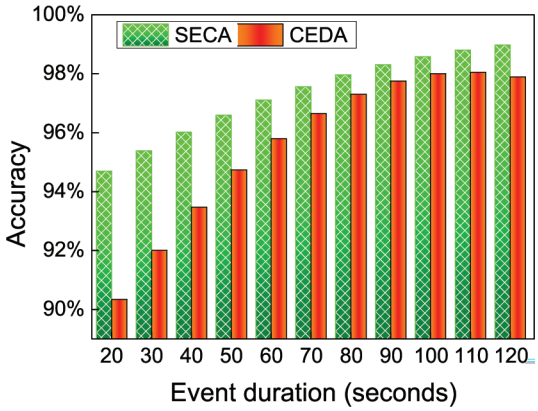


5.4 Detection performance with varying event duration

This section carries out a couple of experiments to evaluate to what extent the event duration influences the detection accuracy of SECA scheme. It also examines the appropriate values of the event duration in given scenarios.

Figure 11 illustrates the experimental results with varying the event duration from 20 to 120 seconds. Both SECA and CEDA schemes are capable of achieving high level of detection accuracy (the least accuracy is bigger than 85%.), which indicates that they are not heavily influenced by event duration. The longer the event duration is, the higher level of detection accuracy SECA and CEDA can achieve. Figure 11 also indicates that SECA is much more accurate than CEDA in handling events with various life span. Consider that SECA gets a high level of detection accuracy when the event duration is in a range between 20 and 50 seconds, we set the event duration as a random value in the same range.

Figure 11 Detection accuracy with varying event duration



5.5 Lessons learned

So far we have presented all evaluations. Experimental results show that SECA is appropriate for context-aware application in asynchronous pervasive computing environments. Particularly, SECA outperforms CEDA regarding concurrent event detection of context inconsistency. It is capable of tolerating fairly long message delay with high level of detection accuracy. Finally, it is robust to the variations of event duration, making it desirable in asynchronous pervasive computing environments where message communication suffers various delays owing to the network limitation.

6 Conclusion

In this paper, we have studied concurrent event detection of context consistency checking in asynchronous ubiquitous computing environments. We have proposed the snapshot timestamp and based on it we have put forward the SECA scheme. Extensive experimental results show that SECA is desirable in context-aware applications and outperforms CEDA in terms of concurrent event detection accuracy and tolerating message delay and event duration.

Currently, SECA scheme could be further improved in the following perspectives. Firstly, we plan to investigate how SECA performs in large-scale pervasive computing environments with thousands of participants. Secondly, we will study whether and how SECA copes with the dynamic changes of processes involved in the concurrent event detection. Finally, we are going to evaluate SECA performances in various scenarios with more types of contexts and consistency constraints.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (Grant Nos. 61103185, 61003247, 61100178, 61073118 and 50905063), the Startup Foundation of Nanjing Normal University (Grant No. 2011119XGQ0072), Natural Science Foundation of the Higher Education Institutions of Jiangsu Province, China (Grant No. 11KJB520009) and the Fundamental Research Funds for the Central Universities (China University of Geosciences at Beijing, China). This work is also supported by Major Program of National Natural Science Foundation of Jiangsu Province (Grant No. BK211005). This research was also supported by the Korea-China Science and Technology Joint Research Center by the National Research Foundation (NRF) under Grant No. 2011-0019905 of Ministry of Education, Science and Technology (MEST), the Korean government.

References

Almeida, P., Baquero, C. and Fonte, V. (2008) 'Interval tree clocks: a logical clock for dynamic systems,' *Princi. Distri. Sys*., *Lecture Notes in Comp. Sci.*, Vol. 5401, pp.259–274.

Bu, Y., Chen, S., Li, J., Tao, X. and Lu, J. (2006) 'Context consistency management using ontology based model', in *Current Trends in Database Technology – EDBT Workshops*, pp.741–755.

Bu, Y., Gu, T., Tao, X., Li, J., Chen, S. and Lu, J. (2006) 'Managing quality of context in pervasive computing', in *Proc. of the 6th. Int. Conf. on Quality Softw*., Washington, DC, USA, pp.193–200.

Chen, Z., Zhang, D., Zhu, R. and Yin, P. (2012) 'A review of automated formal verification of ad hoc routing protocols for wireless sensor networks', *Sensor Letters*, Vol. 6, pp.99–112.

Fidge, C.J. (1988) 'Timestamps in message-passing systems that preserve the partial ordering', in *Proc. of In 11th Austra. Comp. Sci. Conf*., University of Queensland and Griffith University, pp.55–66.

Fujimoto, R.M. (1998) 'Time management in the high level architecture', *Simulation*, vol. 71, No. 6, pp.388–400.

Group, W.H.E.W. (2000) 'Ieee 1516 – standard for modeling and simulation high level architecture framework and rules', *IEEE Standard*, July 2000, pp.1–38.

Gu, T., Wang, X.H., Pung, H.K. and Zhang, D.Q. (2004) 'An ontology-based context model in intelligent environments', in *Proc. of Commu. Netw. Distri. Sys. Model. Simul. Conf.*, 2004, pp.270–275.

Huang, Y., Ma, X., Cao, J., Tao, X. and Lu, J. (2009) 'Concurrent event detection for asynchronous consistency checking of pervasive context', *IEEE International Conference on*

Pervasive Computing and Communications, Galveston, Texas, USA, pp.1–9.

Huang, Y., Yang, Y., Cao, J., Ma, X., Tao, X. and Lu, J. (2011) 'Runtime detection of the concurrency property in asynchronous pervasive computing environments', *IEEE Trans. Paral. Distri. Sys.*, Vol. 23, pp.1–17.

Ji, Y. (2011) 'Performance analysis for indoor location determination', *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 8, Nos. 1/2, pp.3–15.

Jian, M-S., Chou, T-Y. and Hsu, S.H. (2012) 'Mobility corresponding location-aware information services based on embedded rfid platform', *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 9, No. 2, pp.122–131.

Kshemkalyani, A.D. (1996) 'Temporal interactions of intervals in distributed systems', *J. Comp. Sys. Sci.*, Vol. 52, No. 2, pp.287–298.

Lamport, L. (1978) 'Time, clocks and the ordering of events in a distributed system', *Commun. ACM*, Vol. 21, pp.558–565.

Mattern, F. (1994) 'Virtual time and global states of distributed systems', in *Proc. Workshop Paral. Distri. Alg.*, C.M. et al., (Eds.); Elsevier, 1989, pp.215–226, (Reprinted in: Yang, Z. and Marsland, T.A. (Eds.); 'Global States and Time in Distributed Systems', *IEEE*, pp.123–133).

Ni, L., Liu, Y., Lau, Y. and Patil, A. (2004) 'Landmarc: indoor location sensing using active rfid', *Wirel. Netw.*, Vol. 10, No. 6, pp.701–710.

Rao, J., Doraiswamy, S., Thakkar, H. and Colby, L.S. (2006) 'A deferred cleansing method for rfid data analytics', in *Pro. of 32nd Int. Conf. Very Large Data Bases, ser, VLDB 06*, pp.175–186.

Wang, X., Zhang, D., Gu, T. and Pung, H. (2004) 'Ontology based context modeling and reasoning using OWL', in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Orlando, FL, USA: IEEE, March 2004, pp.18–22.

Want, R., Hopper, A., Falcao, V. and Gibbons, J. (1992) 'The active badge location system', *ACM Trans. Info. Sys.*,Vol. 10, pp.91–102.

Xu, C., Cheung, S., Chan, W. and Ye, C. (2008) 'Heuristics-based strategies for resolving context inconsistencies in pervasive computing applications', *Int. Conf. Distri. Comput. Sys.*, Vol. 1, pp.713–721.

Xu, C., Cheung, S.C. and Chan, W.K. (2006) 'Incremental consistency checking for pervasive context', in *Proc. of the 28th Int. Conf. Softw. Eng.* ACM, pp.292–301.

Xu, C., Cheung, S.C., Chan, W.K. and Ye, C. (2010) 'Partial constraint checking for context consistency in pervasive computing', *ACM Trans. Softw. Eng. Methodol.*, Vol. 19, pp.9:1–9:61.

Zhang, D., Chen, M., Huang, H. and Guo, M. (2011) 'Decentralised checking context inconsistency in pervasive computing environments', *The J. Supercomput*, pp.1–18.

Zhang, D., Guo, M., Zhou, J., Kang, D. and Cao, J. (2010) 'Context reasoning using extended evidence theory in pervasive computing environments', *Future Generation Comp. Syst.*, Vol. 26, No. 2, pp.207–216.

Zhang, D., Huang, H., Lai, C-F., Liang, X. and Guo, M. (2011) 'Survey on context-awareness in ubiquitous media', *Multimedia Tools and Applications,* DOI: 10.1007/s11042-011-0940-9, pp.1–33.

Zhang, D., Huang, H., Liao, X. and Chen, M. (2012) 'Empirical study on taxi gps traces for vehicular ad hoc networks', in *Proc. of IEEE International Conference on Communications*, Ottawa, Canada, pp.291–295.

Zhang, D., Wan, J., Liu, Q., Guan, X. and Liang, X. (2012) 'A taxonomy of agent technologies for ubiquitous computing environments', *KSII Transactions on Internet and Information Systems*, Vol. 6, No. 2, pp.547–565.

Zhang, D., Zhou, J., Guo, M., Cao, J. and Li, T. (2011) 'TASA: tag-free activity sensing using RFID tag arrays', *IEEE Trans. on Parallel and Distributed Systems*, Vol. 22, pp.558–570.

Zhang, D., Zou, Q. and Sun, Z. (2012) 'Seca: Snapshot-based event detection for checking asynchronous context consistency in ubiquitous computing', in *Proc. of 2012 IEEE Wireless Communications and Networking Conference*, pp.101–106, Paris, France.

Zhou, Z., Bhiri, S. and Hauswirth, M. (2008) 'Control and data dependencies in business processes based on semantic business activities', in *Proc. of the tenth International Conference on Information Integration and Web-based Applications Services,* 2008, pp.257–263.

Zhou, Z., Bhiri, S., Zhuge, H. and Gaaloul, W. (2012) 'Service protocol adaptability assessment based on novel walk computation', *IEEE Transactions on Systems Man and Cybernetics, Part A: Systems and Humans, DOI: 10.1109/TSMCA.2012.2183362*, pp.1–32.