Chenhua Chen, Chunyang Ye and Hans-Arno Jacobsen, Hybrid Context Inconsistency Resolution for Context-Aware Services, to appear in Proceedings of 9^{th} IEEE International Conference on Pervasive Computing and Communications (PerCom'11), 2011.

Hybrid Context Inconsistency Resolution for Context-Aware Services*

Chenhua Chen
Department of Computer Science
Saarland University
Saarbrücken, Germany
ch.chenhua@googlemail.com

Chunyang Ye†
MSRG, University of Toronto
chunyang@msrg.utoronto.ca
IOS, Chinese Academy of Sciences
cyye@otcaix.iscas.ac.cn

Hans-Arno Jacobsen
Dept. of Elec. & Comp. Eng.
University of Toronto
Toronto, Canada
jacobsen@eecg.toronto.edu

***Abstract*—Context-aware applications automatically adapt their behavior according to environmental conditions, also known as contexts. However, in practice contexts are often inaccurate, noisy or even inconsistent (e.g., two RFID readers may report different numbers for the same set of goods processed). These kinds of problematic contexts may cause context-aware applications to behave abnormally or even fail. It is thus desirable to detect and resolve context inconsistency.**

In this paper, we propose a hybrid approach to detect problematic contexts and resolve resulting context inconsistencies with the help of context-aware application semantics. By combining low-level context inconsistency resolution with high-level application error recovery, our approach can resolve the inconsistent contexts more effectively. Moreover, error recovery cost for context-aware applications is reduced. Our experimental results show that our approach outperforms existing approaches in terms of more accurate inconsistency resolution and less error recovery cost.

***Keywords*-Context-awareness; Inconsistency Resolution; Error Recovery**

I. Introduction

The advent of powerful, embedded computing devices allows applications to become aware of environment changes (e.g., location, temperature). By explicitly modeling environmental conditions as contexts in applications and sensing contexts at runtime via underlying sensors, applications can adapt their behavior automatically when the environment changes. Applications such as these are referred to as *context-aware* applications [29].

Since a context-aware application adapts its behavior automatically based on the context, the quality of the context is extremely important [5, 16, 28]. However, due to inherent noise in the environment in practice, poorly functioning or malfunctioning sensors, or lack of sensitivity, the collected context may be inaccurate, incomplete, or even inconsistent [2, 9, 19, 24, 26]. For example, two RFID readers may report different numbers for the same set of goods processed on a conveyor belt. An inaccurate, incomplete and inconsistent context may cause applications to behave abnormally and deviate from their original specifications [21]. For example, a warehouse management system may fail to register the correct number of goods due to inconsistent values reported by different RFID readers processing the goods.

To address this issue, many researchers suggested filtering or repairing *problematic contexts* (e.g., inaccurate, incomplete or noisy contexts) [2, 8, 19, 26, 27]. These approaches specify context *consistency constraints* to classify contexts into two categories: *consistent* and *inconsistent*. For example, one can define a consistency constraint to verify whether the number of goods reported by different RFID readers is equal. Whenever a context is updated by a sensor, associated context consistency constraints are verified. If a given context violates a consistency constraint, the context is marked as inconsistent. This inconsistency detection is complemented with strategies to filter or repair inconsistent contexts. For example, approaches exist that suggest removing context to keep the consistency constraints satisfied for the remaining contexts [8]. The strategies include removing the latest context, the oldest context, or the least frequently used context, for example. The work in [8, 26] also proposed to repair problematic context sources (e.g., sensors) to reduce potential context inconsistencies. However, these approaches have two major limitations.

First, it is difficult to identify the problematic contexts accurately. The aforementioned strategies to filter and repair contexts are usually based on heuristics. For example, a newly established context may be classified as problematic because the pre-existing contexts satisfied the constraints until the new context came about. These kinds of heuristics lack convincing evidence and may lead to incorrect out-

* This research is supported in part by an Ontario Early Researcher Award, MITACS NCE, and the National Natural Science Foundation of China under Grants No. 60903052, 90718033, 60773028; the National Basic Research Program of China under Grant No.2009CB320704; the National High-Tech Research and Development Plan of China under Grant No. 2007AA010301.

† Corresponding author

comes. As a result, some "*good*" contexts may be removed whereas some problematic contexts are kept. For example, suppose the first RFID reader reports an inaccurate number for the goods scanned on the conveyor, whereas the second reader reports the correct number. The strategy to remove the latest context would filter the correct number reported by the second reader and keep the incorrect one reported by the first reader.

Second, resolution approaches such as these rely heavily on how accurately and completely defined the consistency constraints are. If there are no consistency constraints or the consistency constraints are inaccurate in classifying certain problematic contexts as inconsistent, these problematic contexts will not be filtered or repaired. In practice, consistency constraints may not always be available and completely defined due to the open environments for pervasive computing applications. For example, two RFID readers may report the inaccurate but same total number for a set of goods scanned. The constraint to check whether the reported results are the same is unable to discover this inaccuracy.

Therefore, if the consistency constraints are inaccurate or incomplete, or the context inconsistency resolution does not filter out all the problematic contexts, some problematic contexts may still affect the context-aware application and cause potential failure. For example, the warehouse management system would register the wrong number of goods to be stocked in the warehouse based on the incorrectly reported context by the readers. A subsequent transaction that operates over the reported number of goods would incur an error because of the inaccurate count of goods available in the warehouse.

To overcome these limitations, in this paper we propose a hybrid approach to resolve context inconsistencies. The basic idea is to combine high-level application error recovery with low-level context inconsistency resolution strategies. Fig. 1 illustrates our approach. Let T_0 represent the time that a context inconsistency is detected at the context-aware middleware level and let T_2 represent the time that the context-aware application is found to deviate or fail due to problematic contexts related to the context inconsistency. Our approach postpones the context inconsistency resolution to sometime T_1 between T_0 and T_2. The objective is to make use of additional application semantics gained from the context-aware application during the period between T_0 and T_1 to help identify the problematic contexts more accurately. For example, when the goods are transferred from the conveyor to the shelf in the warehouse, the total weight of the goods reported by the sensors in the shelf and the specification of the type of goods updated by the warehouse management system could be used to estimate the number of goods and correct the inaccurate count in this application.

In addition, our approach applies error recovery methods to restore applications from an error state caused by problematic contexts (undetected or detected, since our approach postpones the resolution). For example, the warehouse management system can apply a backward error recovery approach [13, 30] to roll back the selling transaction and compensate for any side effects (e.g., refund the money) of the transaction; or the warehouse management system can apply a forward error recover approach [13, 30] by revising the transaction and resuming the transaction to operate over the true number of goods in the warehouse.

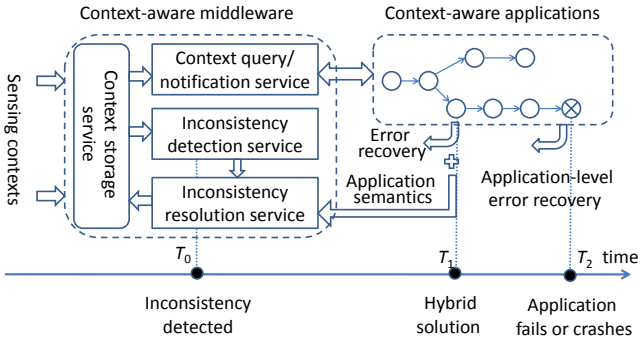


Figure 1: Hybrid solution.

The challenges of this approach are two-fold. (1) making use of the additional semantics of the context-aware applications to identify problematic contexts accurately; and 2) balancing the resolution accuracy and error recovery cost. The resolution shortly after T_0 may be inaccurate due to lack of sufficient application information to identify the problematic contexts. On the other hand, the error recovery at a time close to T_2 may be too late because the error recovery cost may be unacceptable (e.g., the selling transaction may have been confirmed and a service-level penalty would be incurred if the transaction were aborted or modified). To address these challenges, we propose a conflict correlation model to calculate the probability of a context being problematic based on application semantics. We also propose a cost model to trade off the resolution accuracy and error recovery cost. An experimental evaluation using a warehouse management system shows that our approach is promising. Compared to existing work, our approach achieves at least 15% more resolution accuracy and at lease 27% less error recovery cost.

The rest of this paper is organized as follows: Section II introduces our methodology. Section III evaluates our approach empirically. Section IV discusses the limitations of our work and suggests potential improvements. Section V puts our work in the context of related approaches.

II. Methodology

A. Overview

As mentioned in Section I, inconsistent contexts are usually resolved either at the middleware level by approaches that filter contexts based on heuristic strategies, or at the

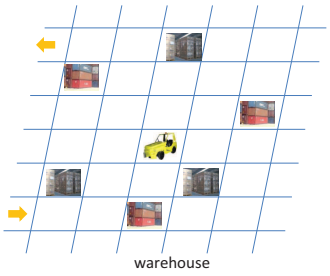


Figure 2: Warehouse management scenario.

application level by error recovery approaches. Existing approaches usually suffer from either low resolution accuracy or unacceptable error recovery cost. In this section, we illustrate our approach to address these problems by developing a novel hybrid approach. Our objectives are to increase the resolution accuracy and reduce the error recovery cost.

The basic idea is to postpone the decision for context inconsistency resolution at the middleware level and to do error recovery earlier at the application level. The benefits are two-fold: (1) application information can be collected during the postponed period to help improve the resolution accuracy for inconsistent contexts and (2) doing error recovery earlier will help to avoid unacceptable error recovery cost for applications.

The key point of this approach is what kinds of information can be collected from the application and when to make the decision to resolve the context inconsistency. We observe that the behavior of context-aware applications is of help in identifying problematic contexts.

Let us illustrate the basic idea using the example in Fig. 2, where an unmanned fork truck picks up goods in a warehouse. The truck is equipped with RFID to identify its current position based on an indoor location sensing service (e.g., LANDMARC [18]). The fork truck is operating autonomously based on location contexts. In each step, the truck can move to either one of four quadrants that are left, right, up or down from its current position given that these positions are not occupied by goods. Due to noise and signal reflection from goods, the location sensing may introduce positioning errors (e.g., the current location is (1, 3) whereas the sensing result gives (4, 6)). As a result, the truck may not be able to complete the job successfully (e.g., may pick up wrong goods).

Suppose the value of the last location context of the truck is (2, 3), and after moving one step, the new sensing value for the location context is (4, 5). According to the consistency constraint, – that is, the distance between two consecutive location contexts should be less than or equal to 1 after moving one step, – there is an inconsistency between these two contexts. The probability of either contexts being problematic is 50%. If we were to apply heuristic strategies (e.g., removing the latest, removing the oldest context etc.) to resolve the inconsistency, the resolution may be inaccurate with a probability of 50%. If the truck continues to move instead of stopping to resolve the inconsistency, it will receive a new location context (let's say (4, 4)). From this information, we can deduce that the location (2, 3) is more likely problematic, because the truck can move from (4, 5) to (4, 4) in one step, but cannot move from (2, 3) to (4, 4) in two steps. Intuitively, given a fixed context error rate, if the truck postpones the inconsistency resolution to a few steps later, it may get new location information to make the resolution more accurate. However, the truck cannot postpone the resolution indefinitely, because this may cause severe errors for the truck (e.g., pick up wrong goods, or get stuck at some location) with the resulting error recovery cost being unacceptable (e.g., truck cannot pick up the required goods in a specified time).

Our solution trades off the resolution accuracy against error recovery cost. In the rest of this section, we present a context-correlation model to calculate the probability of a context being problematic based on the contexts and the behavior of the context-aware application. The context with the highest probability is regarded as the most likely one being problematic. We also propose a cost model to estimate the error recovery cost for the context-aware application. In addition, we develop a resolution algorithm to determine when to resolve context inconsistency with a high accuracy as well as an acceptable error recovery cost.

B. Context-Correlation Model

As mentioned in Section II-A, our solution is to postpone the inconsistency resolution and make use of context-aware application information to increase the resolution accuracy. To do so, we define a correlation model to describe the relationship between contexts before an action of a context-aware application is invoked and contexts after the action is executed based on application semantics. Without loss of generality, we adopt the context model proposed in [20], and define a context as a set of variable-value tuples.

Definition 1. *A context is a set* $\{(x_1, t_1, v_1), \cdots, (x_n, t_n, v_n)\}$*, where* t_i *and* v_i *are the type and the value of variable* x_i*, respectively.*[1]

For example, the location context in Fig. 2 can be represented as $\{(x_1, real, 3.0), (y_1, real, 5.0)\}$, where x_1 and y_1 represent the x-axis and y-axis of the map, respectively. Note that the concept of contexts in our model includes the internal states of an application (e.g., the truck has picked up the goods) and its external environment (e.g., the location of the truck). To simplify the development and maintenance of context-aware applications, context-aware middleware systems have been proposed to manage contexts

[1]Note that the value of a variable can be a concrete value (e.g., $v_i = 2$) or a constraint specifying the range of the value (e.g., $v_i > 2$).

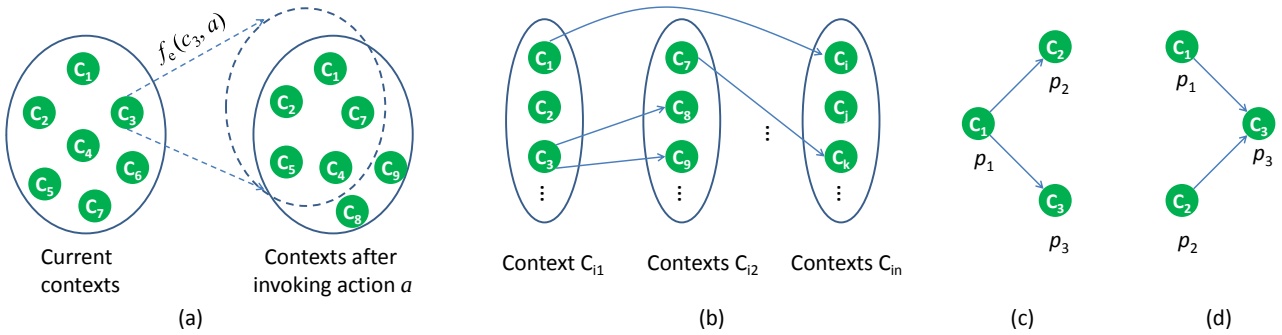


Figure 3: Context correlation and probability model.

for applications [1, 10, 17]. Middleware systems manage, collect, filter, aggregate, and maintain contexts for applications and their underlying sensing devices. Applications can query and make use of contexts through the interfaces provided by the middleware systems. With middleware support, context-aware applications can adapt their behavior automatically by defining rules to handle context changes. The following definition formalizes the model of a context-aware application underlying this paper.

Definition 2. *A context-aware application P is a tuple $\{C, A, R\}$, where C is the set of contexts defined for the application, A is the set of actions (also referred to as tasks) for the application, and $R \equiv C \times A$ is the set of rules.*

Context-aware middleware is responsible for monitoring context changes and invoking rules of applications. Given a rule $(c_1, a) \in R$, the intuitive meaning is that if the underlying middleware detects a change to current context c_1 (e.g., the current location is changed to $c_1 \equiv \{(x_1, real, 4.0), (y_1, real, 5.0)\}$, representing that the truck reaches the destination c_1), the action a (e.g., the truck lifts the goods) is invoked. As explained in Section II-A, the effects of an action in the application may be helpful to identify the problematic contexts. For example, if the truck moves one step, the new location should be within the circle with the last location as the center and the radius equivalent to the length of one step. Otherwise, either the last context or the new context should be problematic. By exploring such application-specific constraints, we can reason about which context is most likely problematic.

To capture the effect of an action on the contexts of an application, Definition 3 introduces an *effect function*, f_e.

Definition 3. *Given a context-aware application P = $\{C, A, R\}$, the effects of actions in the application can be defined as the effect function $f_e : C \times A \rightarrow \{B | B \subseteq C\}$.*

For example, as illustrated in Fig. 3a, $C_2 = f_e(c_3, a)$ describes the effects of moving one step for the truck, where $c_3 \equiv \{(x_1, real, v_1)\ , (y_1, real, v_2)\}$ is its current location and $C_2 \equiv \{\{(x_1, real, v_1'), (y_1, real, v_2')\} | (v_1' - v_1)^2 + (v_2' - v_2)^2 \leq 1\}$. Note that C_2 represents the scope of possible valid contexts after action a is invoked given c_3 is a valid context. Let NC be the set of contexts after action a is invoked. Now, $\forall nc \in NC$, we can compare nc to C_2 to determine whether c_3 is a problematic context or not. If $nc \in C_2$, then it is possible that both c_3 and nc are accurate or inaccurate. However, if $nc \notin C_2$, then at least one of them is inaccurate. Therefore, we have the following implications:

- If $nc \notin f_e(c_3, a)$, and c_3 is accurate, then the probability of nc being inaccurate is 1, that is, $p(nc\ is\ inaccurate \mid c_3\ is\ accurate) = 1$.
- If $nc \notin f_e(c_3, a)$, and nc is accurate, then the probability of c_3 being inaccurate is 1, that is, $p(c_3\ is\ inaccurate \mid nc\ is\ accurate) = 1$.

To describe the relationship between two contexts, we connect two contexts by a line, denoted as $conflict(c_3, nc)$, to represent the correlation between c_3 and nc, that is, $conflict(c_3, nc) \equiv nc \notin f_e(c_3, a)$.

Let us take Fig. 3b as an example. C_{i2} represents the set of contexts after invoking action a. Since c_8 and $c_9 \notin f_e(c_3, a)$, we connect c_3 from C_{i1} and c_8 and c_9 from C_{i2} to represent this correlation relationship. Similarly, suppose C_{i3} is the set of contexts after the application invokes two actions a, b, and $c \in C_{i1}$, $nc \in C_{i3}$, $conflict(c, nc)$ if and only if $\neg\exists\ c' \in f_e(c, a)$ such that $nc \in f_e(c', b)$. In this way, we can derive the correlation relationship between the context after invoking a sequence of actions and the context before the invocation. By linking contexts resulting from every step of action invocation, we can describe the correlation relationship among contexts as a graph.

Based on this graph, we can derive the probability of each current context being inaccurate, and pick the one with the highest value as the problematic one. For example, given two contexts c_1 and c_2 and $conflict(c_1, c_2)$, suppose the probability of c_1 or c_2 being problematic is p_1 and p_2, respectively, then $p_1 + p_2 \geq 1$. This is because $p_2 \geq p(c_2\ is\ problematic\ and\ c_1\ is\ accurate) = p(c_2\ is\ problematic | c_1\ is\ accurate) \times p(c_1\ is\ accurate) = 1 \times (1 - p_1)$.

However, if there exists another context c_3 such that $conflict(c_1, c_3)$, and the probability of c_3 being problematic is p_3, as illustrated in Fig. 3c, then c_1 is

problematic given either c_2 or c_3 is accurate. Therefore, $p_1 \geq p(either\ c_2\ or\ c_3\ is\ accurate) = 1 - p(both\ c_2\ and\ c_3\ are\ inaccurate) = 1 - p_2 \times p_3$.

Similarly, in another scenario, as illustrated in Fig. 3d, if $conflict(c_1, c_3)$ and $conflict(c_2, c_3)$, then c_3 is problematic given either c_1 or c_2 is accurate. Therefore, $p_3 \geq p(either\ c_1\ or\ c_2\ is\ accurate) = 1 - p_1 \times p_2$.

Based on the above discussion, we obtain the following formula to calculate the probability of each context being problematic.

$$\forall c_0 \in C,\ if\ \exists c_i (i = 1..n) \in C :\ conflict(c_0, c_i)\ or\ conflict(c_i, c_0), then\ p_0 \geq 1 - \prod_{i=1}^{n} p_i. \quad (1)$$

Given this formula, for every context we can apply computations via mathematics packages, such as Algebrator[2] or Matlab[3] to analyze the probability of each context being problematic. Given a set of inconsistent contexts, C', we can choose to resolve the contexts in C' which are most likely problematic.[4]

Intuitively, the resolution should be postponed to a time after many action invocations, where the resolution should be more accurate because more information can be collected to help determine problematic contexts. However, this may also increase the error recovery cost for the application. In Section II-C, we propose a solution to resolve the inconsistency and trade off the accuracy of resolution against the error recovery cost.

C. Inconsistency Resolution

As explained in Section I, when an inconsistency is detected among a set of contexts, our approach does not resolve the inconsistency immediately. Instead, we resume the execution of the application and postpone the inconsistency resolution to a time after some actions have been invoked by the application. In this way, we can use application level semantics (that is, the effects of actions) to correlate contexts between different action steps and make use of this information to help identify inaccurate contexts in the original set of inconsistent contexts more accurately. After identifying the contexts that are most likely problematic, the inconsistency resolution removes these contexts and recovers the application from the error state caused by problematic contexts. Basically, there are two ways to recover an application from an error state:

Backward Recovery: This method compensates the effects of invoked actions, restores the application to a state free of errors, and then resumes the application from the error-free state. For example, as illustrated in Fig. 4, the problematic context c is sensed at the state s_1 of the application, and the inconsistency caused by c is detected at state s_2' (the dashed line represents that the application is affected by this problematic context c). At state s_3', context c is identified to be problematic, and the inconsistency resolution is applied. The application removes context c and recovers from state s_3' to s_1 using a backward recovery approach. That is, the application invokes action b_3 and b_2 to compensate the effects of a_3 and a_2, respectively. Then, the application can resume the execution from state s_1.

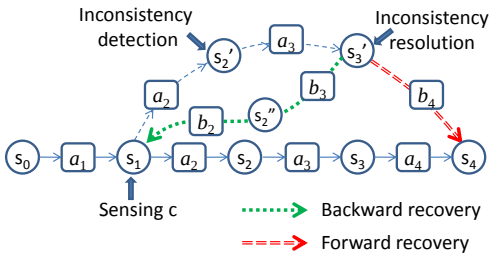


Figure 4: Backward and forward error recovery.

Forward Recovery: This strategy allows the application to execute a sequence of actions and to transfer from the current error state to a state free of errors. For example, as illustrated in Fig. 4, instead of compensating the invoked actions, the application invokes another action b_4 at the error state s_3' and transfers to an error-free state s_4.

In practice, both backward and forward error recovery approaches can be applied in concert. For example, the truck in the warehouse may first apply backward recovery to backtrack to a position from the current position (because the truck is stuck at this position) and then apply forward recovery by choosing another path from the new position. How an application chooses to recover from an error state is determined by its recovery plan, which is specific to the application and designed by the application developers. How to develop a recovery plan for an application is out of scope of this paper.

Since our approach applies high-level application semantics to help identify problematic contexts, in general, the more application information we have, the higher the possibility that the problematic contexts can be identified. However, the inconsistency resolution cannot be postponed indefinitely, because this may cause unacceptable error recovery cost for the application. Therefore, when to resolve the context inconsistency is a crucial issue.

To address this issue, we propose a cost model for context-aware applications to evaluate the error recovery cost. In this model, each action is assigned two costs:

Compensation cost: If backward recovery is applied, the application needs to compensate some invoked actions. The compensation cost for each action a is denoted as

[2]Softmath, Algebrator, 1990s, http://www.softmath.com/

[3]Mathworks, Mathlab, http://www.mathworks.com/products/matlab/

[4]There are different strategies to define the "most likely problematic" contexts. For example, one may choose to resolve the smallest set of problematic contexts that satisfy the aforementioned inequality constraints. In this paper, we use another strategy, selecting the contexts with the highest probability as being problematic among all the possible solutions given the aforementioned inequality constraints.

$cpc(a)$. Note that the cost for each action is determined by application semantics. For example, if invoking an action has no side effects for an application, then the compensation cost for the action is zero. Instead, if the side effects of an action cannot be compensated (e.g., the truck moves into a path from where it is not allowed to backtrack), the compensation cost is set to be infinite.

Execution cost: If forward recovery is applied, the application needs to invoke some actions to transfer its state to an error-free state. The execution cost for invoking each action a is denoted as $ecc(a)$. Note that the execution cost for each action is also determined by application semantics. For example, the execution cost may include the time needed, the resource consumed etcetera.

Given the compensation cost and execution cost for each action, we can calculate the cost needed for each recovery plan. If the application applies backward recovery to compensate a sequence of invoked actions $a_1, a_2, \cdots, a_n$, then the cost for this recovery plan is equal to $\sum_{i=1}^{n} cpc(a_i)$. If the application applies forward recovery by invoking a sequence of actions $b_1, b_2, \cdots, b_m$ to recover from errors, the cost for this recovery plan is equal to $\sum_{i=i}^{m} ece(b_i)$. If a hybrid plan is taken, that is, the application compensates a sequence of invoked actions $a_1, a_2, \cdots, a_n$ first and then invokes a sequence of actions $b_1, b_2, \cdots, b_m$ for forward recovery of errors, the cost for the hybrid plan is equal to $\sum_{i=1}^{n} cpc(a_i) + \sum_{j=1}^{m} ece(b_j)$.

Algorithm 1 Hybrid context inconsistency resolution.

Input:
A context-aware service P;
Output:
The set of problematic contexts and error recovery plan;
1: let CC be the set of current contexts;
2: **while** $not\ terminated$ **do**
3: set PC equal to CC; /*PC means previous contexts*/
4: invoke next action a ;
5: let CC be the set of current contexts;
6: **for** $\forall cc \in PC, nc \in CC$ **do**
7: **if** $nc \notin f_e(cc, a)$ **then**
8: link cc and nc
9: **end if**
10: **end for**
11: calculate the probability of each context being problematic;
12: calculate the cost for every recovery plan;
13: **end while**
14: let IC be the set of contexts with the highest probability;
15: remove IC from the context sets;
16: let ERP be the error recovery plan with the lowest cost;
17: execute the error recovery plan ERP.

Given the recovery cost, we propose an algorithm to resolve the context inconsistency, as illustrated in Algorithm 1. The basic idea is to calculate the probability of each context being problematic, given additional information after invoking an action, and then calculate the cost for each recovery plan. The application will continue to postpone the inconsistency resolution until the termination condition is satisfied (Line 2).

In the termination condition, the recovery cost may be set to a value larger than an unacceptable value, or the probability of a context being problematic is set to a value larger than a reasonable threshold. The application picks the context with largest probability of being problematic to resolve, and selects the error recovery plan with the lowest cost to restore the application to an error-free state (Lines 14 to 17).

III. Evaluation

This section evaluates the benefits of our approach quantitatively in terms of accuracy of inconsistency resolution and error recovery cost as well as scalability.

A. Experimental Setup

In our experiment, three approaches are investigated. Two approaches are used as baseline: One is to apply only low-level context inconsistency resolution by using various strategies (that is, remove the latest context, represented as L-RL and remove the oldest context, represented as L-RO). Another one is to apply only high-level application error recovery. No context inconsistency resolution capability is provided to the application. Instead, the application recovers from the errors caused by inconsistent contexts using application level error recovery plans. This approach is represented as H-ER. Our approach applies a hybrid solution to combine low-level context inconsistency resolution and application level error recovery, denoted as **M-H**. We compare our approach to the other approaches in terms of accuracy of inconsistency resolution and error recovery cost.

The warehouse application scenario introduced in Section II-A is used in our experimental evaluation. The map of the warehouse is set to 16×16 quadrants and goods are randomly placed in some quadrants. In one step, the truck can move towards either of the four quadrants that are left, right, up or down from its current position given these quadrants are not occupied by goods. In the experiment, the truck receives contexts from the underlying middleware with a given inaccuracy rate. The location context used in this application is represented as $(x, int, v_x), (y, int, v_y)$, where x and y represent the x-axis and y-axis, respectively, and $v_x, v_y \in [1-16]$. Another type of context is the information about occupied quadrants near the truck. This information is used by the truck to decide how to move in the next step.

The truck applies a heuristic to search the target goods following the strategy of moving toward the next quadrant closest to its current location with the shortest distance to the

target good. To evaluate the error recovery cost, we count the number of steps required to pick up the target goods by the truck for each approach.

The more steps are needed, the higher the error recovery cost is in the approach. If the truck cannot pick up the goods, a penalty is added to the cost. To evaluate the accuracy of inconsistency resolution, we count the number of resolutions and the number of correct resolutions by each approach. The accuracy rate is regarded as the ratio of the number of correct resolutions over the total number of resolutions. A higher ratio means that the approach has a higher accuracy for inconsistency resolution.

B. Data Analysis

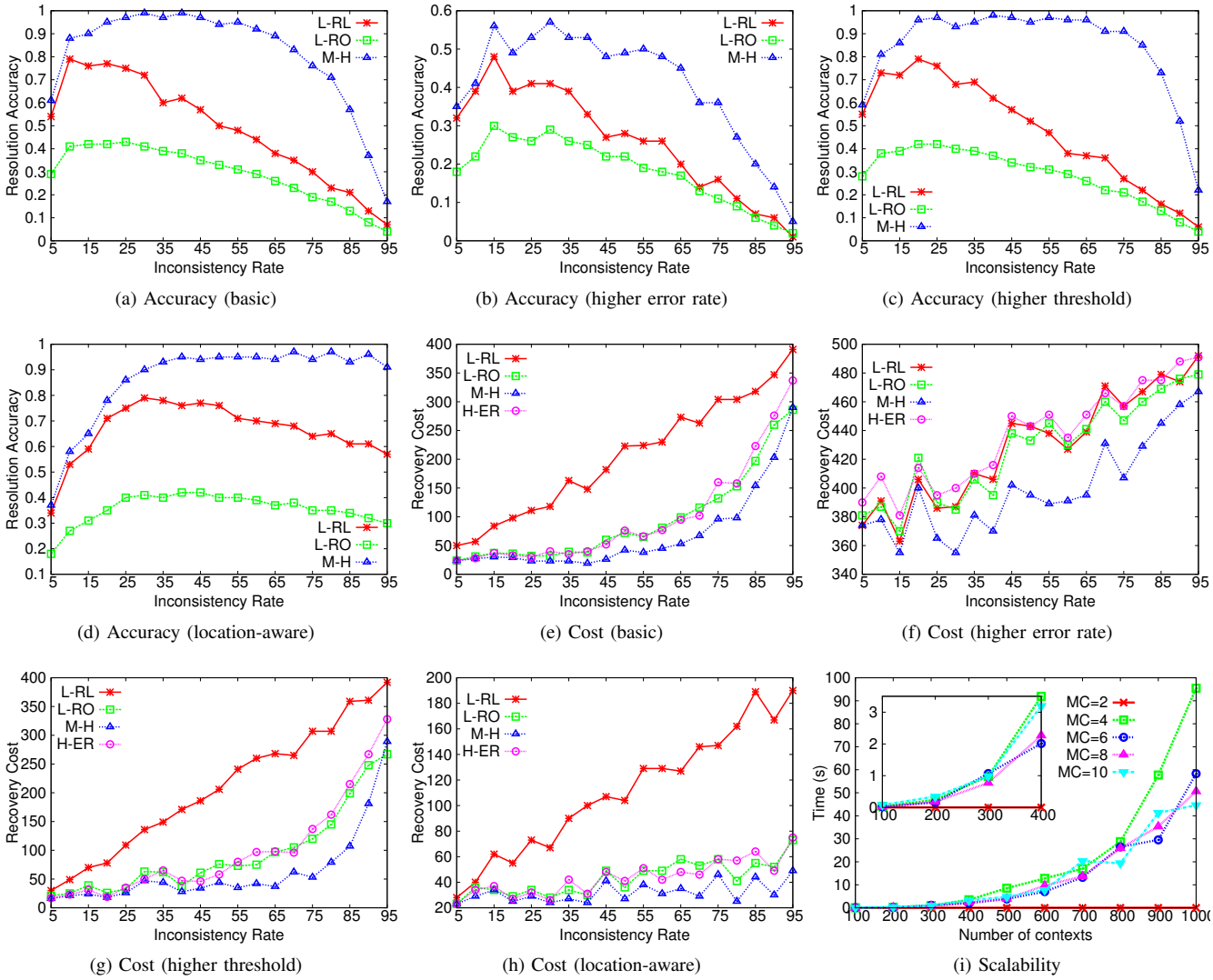
In the experiment, we evaluated and compared the resolution accuracy of our approach and the two low-level resolution approaches (that is, L-RL and L-RO). We fixed the ratio of problematic contexts and applied the three solutions to the application, respectively. We repeated the experiment by varying the ratio of problematic contexts from 5% to 95%. The experiment is repeated 1000 times and the average result is shown in Fig. 5a. On average, our approach used 13 historical contexts. From the experimental results, we can observe that the accuracy rate of L-RL is higher than that of L-RO in this application, but the accuracy rate of our approach is much higher than both low-level approaches. We can also observe that when the ratio of problematic contexts is much higher (e.g., 90%), the accuracy rates of all these approaches drop dramatically. This is because with more problematic contexts, all the solutions cannot determine which contexts are accurate, nor are they able to resolve those problematic contexts correctly. We can observe that even though the ratio of problematic contexts is high, our solution still outperforms the L-RL and L-RO solutions. We also observed that when the inconsistency rate is small (e.g., $\leq$10%), the resolution accuracy rate increases along with the increase of the inconsistency rate for all the solutions. This is because when the number of inaccurate contexts is small, increasing the proportion of inaccurate contexts increases the chance of an inaccurate context being selected in the resolution, no matter which strategy is applied.

In the experiment, some factors may threaten the validity of the experimental results, such as the error rate of the application (that is, the sensitivity of the application to behave incorrectly for problematic contexts) and the threshold of the probability of a context being problematic to resolve. Therefore, we repeated the experiment by varying these factors to evaluate their impacts. Fig. 5b shows the result with a higher error rate.[5] The result shows that our approach also performs better than the other approaches when the error rate is high. We then changed the threshold to a higher value and repeated the experiment. The result shows that our approach has a higher resolution accuracy with a higher threshold, as illustrated in Fig. 5c. In addition, the ratio of problematic contexts in practice may be different in different locations. We then changed the ratio of problematic contexts for different quadrants in the warehouse map and repeated the experiment. We varied the average ratio of problematic contexts in these quadrants from 5% to 95%. The result is shown in Fig. 5d. Compared to the quadrants with the same ratios of problematic contexts, all the solutions have a higher resolution accuracy even though the rate of problematic contexts is high (e.g., 90%). This may be because the problematic contexts are grouped to nearby locations surrounded by accurate contexts that increase the resolution accuracy. We also observed that our approach performs better than the existing approaches in this scenario.

In the experiment, we also recorded the recovery costs for all solutions. We fixed the ratio of problematic contexts and applied all the aforementioned solutions to the application, respectively. We repeated the experiment by varying the ratio of problematic contexts from 5% to 95%. The average result of 1000 repeated runs is shown in Fig. 5e. From the results, we observe that our approach has the lowest recovery cost among all solutions. We also repeated the experiment with a higher error rate, a higher threshold and various problematic context rates for different quadrants. The results are shown in Fig. 5f, Fig. 5g, and Fig. 5h, respectively. In all these scenarios, our solution outperforms the other solutions with a lower recovery cost. We also observed that even though the L-RL solution has a higher resolution accuracy rate than the L-RO solution, its error recovery cost is higher than that of L-RO. This is because the penalty of inaccurate resolution is low in this application setting or is even helpful to restrict the search paths for the target goods. As a result, the cost of the L-RO solution is lower than that of L-RL. However, when we increased the error rate for the application (shown in Fig. 5f), we can observe that the L-RL solution performs as good as (in some cases even better than) the L-RO solution.

In addition, we randomly generated context conflict graphs with varying number of contexts (from 100 to 1000) and varying number of maximum conflicts per context (from 2 to 10, denoted as MC2 to MC10). We then ran our solution on a PC with a 2GHz CPU and 1GB memory to calculate the most likely problematic contexts. The experiment was repeated 100 times and the average time needed was recorded. The result is shown in Fig. 5i. The result shows that the time needed is quite small (i.e., less than 100 second for up to 1000 contexts). We can also observe that the more conflicts per context, the less time is needed (except MC2, which is a special and simpler case). This is because more conflicts mean more inequality constraints and a smaller solution space.

[5] In the experiment, we simulated a higher error rate by increasing the number of goods in the warehouse. The more quadrants are occupied by goods, the fewer paths are available for the truck to reach the target good; therefore, the application is more likely to behave incorrectly (e.g., got stuck).



(a) Accuracy (basic) (b) Accuracy (higher error rate) (c) Accuracy (higher threshold)

(d) Accuracy (location-aware) (e) Cost (basic) (f) Cost (higher error rate)

(g) Cost (higher threshold) (h) Cost (location-aware) (i) Scalability

Figure 5: Accuracy rate, error recovery cost and scalability of inconsistency resolution.

IV. Discussions

In this section, we discuss the limitations of our current work and suggest potential improvements.

Maximum likelihood estimation. Maximum likelihood estimation (MLE) [7] is a well-known statistical method to estimate a statistical model's parameters such that the model can best fit the observed data. The general idea behind MLE can be applied to address the issues of determining the reasons to explain the observed phenomena. For example, MLE has been applied to diagnose faults in computer networks [11, 22, 23]. By specifying the probability correlation among faults and symptoms in computer networks, the set of faults that can best explain the observed symptoms are marked as the root cause of a failure. Our approach follows a similar idea to identify the contexts that are most likely problematic. The difference is that we correlate conflicting contexts based on the semantics of context-aware applications instead of cause-effect relationships between contexts. It is unknown whether a cause-effect relationship between contexts exist or not. We are not determining the reason behind inaccurate contexts but estimate which contexts are more likely inaccurate based on conflicting constraints. The limitation of MLE is that the estimated results can be heavily biased for a small sample set. To address this issue, we postpone the inconsistency resolution to collect as much application semantics as possible to estimate the problematic contexts with an acceptable error recovery cost.

Sensitivity of effect functions. In our approach, we employ an effect function to describe the semantics of each action and correlate contexts in different steps of action invocation. In general, if the effect function for each action is more sensitive, that is, it is more effective to exclude problematic contexts after invoking an action from an accurate context, then the more precise our approach can resolve

problematic contexts. This is because we can gain more conflict-correlation information to derive the probability for each context being inaccurate. How to design more sensitive effect functions is out of the scope of this paper. This may be determined by the semantics of applications. We plan to investigate the factors that effect the sensitivity of the effect function in future work.

Correlation of contexts. In our approach, the effect function correlates an individual context to a set of contexts after an action is invoked. In practice, some contexts may be correlated and the effect function may not be designed for a single context, but for some of them jointly. To apply our approach in these cases, we may need to revise our context-correlation model to map contexts from n to m instead of 1 to m. Similarly, we may also need to revise the possibility model to reflect such correlation-relationships. Another possible extension is to assign a value between 0 and 1 to each conflict relationship between two contexts in the correlation graph, indicating the confidence in the conflict relationship. We plan to extend our work to handle such scenarios in future work.

Fault-tolerant context management. One possible way to reduce inconsistent contexts is to apply fault-tolerant techniques based on redundancy in context management. For example, a warehouse management system may deploy more than one RFID reader to read the same set of RFID tags. By comparing the results from different readers, the system may be able to filter out inconsistent data points. However, as mentioned in Section I, redundant RFID readers may fail to detect the inconsistent contexts if all readers report inaccurate but identical results due impact attributable to environmental noise, for example. Another solution is to apply different types of devices to sense the same context. For example, a location context could be calculated based on the results of GPS devices as well as cameras. This solution can help to avoid the same bias in the reported results by different devices, but the deployment of different devices increases not only the cost but also the complexity of context management. In contrast, our solution applied high-level application semantics to help identify inaccurate contexts without the need for redundant devices. Our approach is orthogonal to approaches based on redundancy.

Handling different types of contexts. In principle, our approach can handle multiple different types of contexts at the same time because our correlation and probability model is not restricted to a single type of context information. For example, temperature contexts reported by a car engine can be correlated to identify the inaccurate contexts based on the power output and work mode of the engine. Time contexts can be correlated according to the duration of tasks. Weight contexts can be correlated based on the updates of goods in the warehouse and their specifications, etcetera.

V. Related Work

In this section, we review recent related work in the area of context-aware application support. We summarize the state-of-the-art on context inconsistency detection and resolution, testing context-aware applications, and middleware support for context-aware applications.

Inconsistency detection. Due to the dynamic and noisy nature of environments, contexts are usually full of noise and errors that let applications processing them behave abnormally. To address this problem, much research has been devoted to detecting context inconsistency. For example, Xu *et al.* [28] proposed an incremental approach to detect context inconsistency efficiently by reusing detection results. Jeffery *et al.* [9] proposed a statistical model to detect and filter inaccurate RFID data. Sama *et al.* [21] proposed a static approach to analyze and detect faults in context-aware applications based on a formal model. These approaches can be used as a basis upon which context inconsistency resolution techniques can build.

Inconsistency resolution. Besides detecting context inconsistency, much effort has been devoted to resolve inconsistent contexts. Bu *et al.* [2] proposed to filter inconsistent contexts except the latest one. Heckmann [8] suggested different strategies (e.g., remove latest context, oldest one, the least frequently used one, and so on). These approaches may be inaccurate because these strategies are based on some assumption that may not always hold in practice. Insuk *et al.* [19] proposed an approach to resolve inconsistent contexts based on user preferences. Xu *et al.* proposed an approach to identify inconsistent contexts based on heuristics and to resolve them based on the impact on the application. That is, the contexts that trigger less rules defined by the applications are filtered [27]. Our approach also uses information from context-aware applications to help resolve the inconsistent contexts. The difference is that our approach postpones the resolution to make use of future application semantics and context information to help identify inconsistent contexts. We also apply a cost model to integrate the low-level context inconsistency resolution and high-level application error recovery. In addition, Kulkarni *et al.* [12] proposed a programming model based on exception handling to recover context-aware applications from failures in a forward processing manner. However, the low-level inconsistency resolution and error recovery cost are not addressed in their approach.

Testing context-aware applications. Since context-aware applications may fail to behave as expected due to the integration of underlying contexts, many researchers proposed approaches to test context-aware applications. Lu *et al.* [14, 15] proposed a data-flow approach to analyze the Definition-Use (D-U) of contexts and defined different coverage criteria based on the D-U of contexts to generate and select test cases. Wang *et al.* [25] identified the context-

aware programming points of a context-aware program and explored possible scenarios of context interference in a program. A test case generation approach is then proposed to generate and select test cases to cover these scenarios. In addition, Chan *et al.* [4] proposed to test context-aware programs using metamorphic testing. Our approach also addresses identifying the faults that are caused by inconsistent contexts. The difference is that we apply a model-based approach to identify and resolve inconsistent contexts and eliminate their corresponding impacts on applications.

Context-aware middleware. Recently, many context-aware middleware has been proposed to aid in the development of context-aware applications. For example, Context Toolkit [6] introduces context widgets to collect, synthesize and interpret contexts to decouple the handling and usage of contexts for context-aware applications. Lime [17] and Egospace [10] are context-aware middleware supporting both logical and physical mobility of agents in a context-aware environment with the help of a tuple space. In addition, Bellavista *et al.* [1] proposed a middleware to manage resources in wireless sensor networks. Capra *et al.* [3] resolved profile conflicts among applications at the middleware level based on an auction among partners. Our approach can also be integrated into middleware as a resolution service. The difference between our approach and existing middleware services is that our approach applies a hybrid strategy to resolve the context inconsistency concerning both low-level accuracy and high-level error recovery cost.

VI. Conclusions

In this paper, we present a hybrid solution to resolve context inconsistency for context-aware applications. Our approach combines low-level context inconsistency resolution and high-level application error recovery. By postponing the inconsistency resolution, our approach makes use of information available from the context-aware application to help identify problematic contexts. Moreover, a cost model is proposed to trade off the resolution accuracy of context inconsistency and error recovery costs. An algorithm to resolve context inconsistency is proposed. The experiments show that our approach outperforms existing approaches in terms of resolution accuracy of context inconsistency and error recovery cost for context-aware applications.

In the future work, we plan to conduct more case studies and experiments using real-life applications to investigate the wider applicability of our solution.

Acknowledgment

The authors would like to thank the shepherd and the anonymous reviewers for their useful comments and suggestions to improve the work.

References

[1] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. Context-aware middleware for resource management in the wireless internet. *IEEE Trans. Softw. Eng.*, 29(12):1086–1099, 2003.

[2] Y. Bu, T. Gu, X. Tao, J. Li, S. Chen, and J. Lu. Managing quality of context in pervasive computing. In *QSIC '06*, pages 193–200, Washington, DC, USA, 2006. IEEE Computer Society.

[3] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. Softw. Eng.*, 29(10):929–945, 2003.

[4] W. K. Chan, T. Y. Chen, H. Lu, T. H. Tse, and S. S. Yau. Integration testing of context-sensitive middleware-based applications: a metamorphic approach. *International Journal of Software Engineering and Knowledge Engineering*, 16(5):677–704, 2006.

[5] P. Damián-Reyes, J. Favela, and J. Contreras-Castillo. Uncertainty management in context-aware applications: Increasing usability and user trust. *Wirel. Pers. Commun.*, 56:37–53, 2011.

[6] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16:97–166, December 2001.

[7] R. Fisher. Theory of statistical estimation. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 22, pages 700–725. Cambridge Univ Press, 1925.

[8] D. Heckmann. Situation modeling and smart context retrieval with semantic web technology and conflict resolution. In *IJCAI-MRC'05)*, pages 34–47, Heidelberg, Berlin, 2005. Springer.

[9] S. R. Jeffery, M. Garofalakis, and M. J. Franklin. Adaptive cleaning for rfid data streams. In *VLDB '06*, pages 163–174. VLDB Endowment, 2006.

[10] C. Julien and G.-C. Roman. Egospaces: Facilitating rapid development of context-aware mobile applications. *IEEE Trans. Softw. Eng.*, 32(5):281–298, 2006.

[11] S. Klinger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. A coding approach to event correlation. In *IM'95*, pages 266–277, London, UK, UK, 1995. Chapman & Hall, Ltd.

[12] D. Kulkarni and A. Tripathi. A framework for programming robust context-aware applications. *IEEE Trans. Softw. Eng.*, 36:184–197, 2010.

[13] P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1990.

[14] H. Lu, W. Chan, and T. Tse. Testing pervasive software in the presence of context inconsistency resolution services. In *ICSE '08*, pages 61–70, NY, USA, 2008. ACM.

[15] H. Lu, W. K. Chan, and T. H. Tse. Testing context-aware middleware-centric programs: a data flow approach and an rfid-based experimentation. In *SIGSOFT '06/FSE-14*, pages 242–252, USA, 2006. ACM.

[16] A. Manzoor, H.-L. Truong, and S. Dustdar. Using quality of context to resolve conflicts in context-aware systems. QuaCon'09, pages 144–155, Berlin, Heidelberg, 2009. Springer-Verlag.

[17] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Softw. Eng. Methodol.*, 15(3):279–328, 2006.

[18] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil. Landmarc: indoor location sensing using active rfid. *Wirel. Netw.*, 10(6):701–710, 2004.

[19] I. Park, D. Lee, and S. J. Hyun. A dynamic context-conflict management scheme for group-aware ubiquitous computing environments. In *COMPSAC '05*, pages 359–364, Washington, DC, USA, 2005. IEEE Computer Society.

[20] G.-C. Roman, C. Julien, and J. Payton. A formal treatment of context-awareness. In *FASE'04*, volume 2984 of *LNCS*, pages 12–36. Springer, 2004.

[21] M. Sama, S. Elbaum, F. Raimondi, D. S. Rosenblum, and Z. Wang. Context-aware adaptive applications: Fault patterns and their automated identification. *IEEE Trans. Softw. Eng.*, 36:644–661, 2010.

[22] M. Steinder and A. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, 2004.

[23] M. Steinder and A. S. Sethi. Probabilistic fault diagnosis in communication systems through incremental hypothesis updating. *Comput. Netw.*, 45:537–562, July 2004.

[24] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004, Nottingham/England*, 2004.

[25] Z. Wang, S. Elbaum, and D. S. Rosenblum. Automated generation of context-aware tests. In *ICSE '07*, pages 406–415, Washington, DC, USA, 2007. IEEE Computer Society.

[26] C. Xu and S. C. Cheung. Inconsistency detection and resolution for context-aware middleware support. In *ESEC/FSE-13*, pages 336–345, New York, NY, USA, 2005. ACM.

[27] C. Xu, S. C. Cheung, W. K. Chan, and C. Ye. Heuristics-based strategies for resolving context inconsistencies in pervasive computing applications. In *ICDCS '08*, pages 713–721, Washington, DC, USA, 2008. IEEE Computer Society.

[28] C. Xu, S. C. Cheung, W. K. Chan, and C. Ye. Partial constraint checking for context consistency in pervasive computing. *ACM Trans. Softw. Eng. Methodol.*, 19(3):1–61, 2010.

[29] S. S. Yau, Y. Wang, and F. Karim. Development of situation-aware application software for ubiquitous computing environment. In *COMPSAC '02*, pages 233–238, Washington, DC, USA, 2002. IEEE Computer Society.

[30] C. Ye, S. C. Cheung, W. K. Chan, and C. Xu. Atomicity analysis of service composition across organizations. *IEEE Trans. Softw. Eng.*, 35(1):2–28, 2009.