On Impact-Oriented Automatic Resolution of Pervasive Context Inconsistency

Chang Xu, S.C. Cheung
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Kowloon, Hong Kong, China
{changxu, scc}@cse.ust.hk

W.K. Chan
Department of Computer Science
City University of Hong Kong
Kowloon, Hong Kong, China
wkchan@cs.cityu.edu.hk

Chunyang Ye
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Kowloon, Hong Kong, China
cyye@cse.ust.hk

ABSTRACT
Context-awareness is a capability that allows applications in pervasive computing to adapt themselves continuously to changing contexts of their environments. However, contexts from physical environments may be inconsistent. It affects the correctness of these applications. Existing resolution strategies for context inconsistency have diverse adverse impacts on the context-awareness of applications, such as feeding different amounts of contexts to the applications. In this paper, we examine the impacts of inconsistency resolution and study the extent to which their effects on context-awareness can be reduced. We conduct simulation experiments of two pervasive computing applications. The experimental results show that existing inconsistency resolution strategies adversely affect the context-awareness of applications. This motivates the importance of deploying an impact-oriented approach to respect context-awareness in inconsistency resolution.

Categories and Subject Descriptors
D.2.4 [**Software Engineering**]: Software/Program Verification – *Validation*.

General Terms
Measurement, Performance, Verification

Keywords
Inconsistency Resolution, Pervasive Computing

1. INTRODUCTION
With the advancement of radio frequency, sensor network, and wireless technologies, pervasive computing is increasingly popular. Applications in pervasive computing deliver adaptive services according to changes to their environments. *Contexts* are pieces of information describing an application's dynamic environment. An application's capability to adapt to its changing environment is referred to as *context-awareness*.

Common examples of contexts include object locations, time, light intensity, and ambient temperatures. To ease the use of these contexts in pervasive computing, various application frameworks or middleware infrastructures [2][7][8] have been proposed. For example, application queries are sent to databases to retrieve contexts [11]; contextual conditions are specified and checked to invoke certain application components [4].

ESEC-FSE'07, September 3–7, 2007, Cavat near Dubrovnik, Croatia.
ACM 978-1-59593-812-1/07/0009.

Since context-aware applications rely on their underlying contexts for correct computation, the quality of these contexts is important to the applications. However, contexts in pervasive computing are error-prone due to their environmental nature [1]. As a result, contexts are subject to *inconsistency* (i.e., conflicting with each other). Unexpected outcomes can occur when applications use these inconsistent contexts for computation.

Various strategies for resolving inconsistency have been studied. The drop-latest strategy discards the latest context if it causes any inconsistency with earlier contexts [3]. The drop-all strategy simply discards all contexts that cause any inconsistency [1]. A user-specified strategy allows users to decide how to handle an inconsistency [13]. Automated resolution strategies are desirable in pervasive computing because resolving inconsistency manually is costly and inefficient. The above-mentioned drop-latest and drop-all strategies are two examples of automated resolution strategies. They are simple to implement, but ignore potential impacts on the context-awareness of applications that use these contexts.

Let us illustrate this with the Call Forwarding [12] application that automatically forwards an incoming call to its callee using the nearest phone. The location contexts of all users are managed carefully to avoid any location inconsistency (e.g., a person appears in two different places at the same time). Suppose that there are two location contexts L_{work} and L_{res} indicating that Peter is in the workshop and that he is in another zone restricting the use of any mobile phone, respectively. If the workshop is located outside the restricted zone, the two contexts are inconsistent. To resolve this by dropping contexts, we may discard (1) L_{work} only, (2) L_{res} only, or (3) both L_{work} and L_{res}.

To investigate different impacts of the three alternatives on the application, we assume that Call Forwarding is configured to forward a call only when its target person is in a mobile phone restricted zone. This imposes a requirement to collect continuously the location contexts of a person if there is any context showing that this person is in a restricted zone. We assume that L_{res} is the only location context about Peter entering the restricted zone.

We proceed to examine the above three alternatives. Discarding L_{res} only (or both L_{work} and L_{res} alike) would cause no new location contexts about Peter collected. This is because the only newly acquired location context L_{res} about Peter entering the restricted zone is discarded. As a result, Call Forwarding does not work for Peter according to the assumed configuration. On the other hand, if we choose to discard L_{work} only, the remaining L_{res} shows that

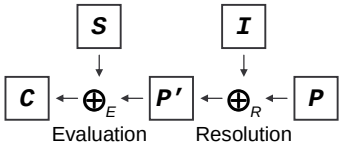

Figure 1. Problem abstraction

Peter is in the restricted zone. Therefore, Call Forwarding continuously collects Peter's location contexts, and diverts a call to Peter if any.

When Peter is in the restricted zone, the use of contexts in the third alternative is desirable because it has the same effect as if no inconsistency resolution had taken place. On the contrary, the context-awareness of the application is adversely affected by the first two alternatives because incoming calls cannot be automatically forwarded to Peter without his location contexts collected.

The example shows the need for inconsistency resolution to consider potential ways of using contexts by an application and the impact on its context-awareness. The function of an application may be unexpectedly affected (e.g., forwarding a call or not) by inconsistency resolution. This perspective has not been well studied in the existing works. We thus formulate two research questions to study in this paper:

1. Can the impact of a resolution strategy on context-awareness be easily calculated?
2. Do we have any useful heuristics for facilitating the calculation of some impact metrics?

The rest of this paper is organized as follows. Section 2 reviews related work in recent years. Section 3 analyzes the challenges in building an impact-oriented resolution framework to respect context-awareness. Section 4 presents our preliminary simulation experiments on two context-aware applications. Finally, Section 5 concludes the paper.

2. Related Work

Context management for pervasive computing is receiving research attention. EgoSpaces [7], Lime [8], and Cabot [13] have tackled different related issues on the management of access control models, transiently shared data, and application contexts in pervasive environments.

We focus on automatic inconsistency resolution. Some related works addressed the context inconsistency problem and proposed similar resolution strategies. Bu et al. [1] suggested discarding all conflicting contexts except the latest one. Insuk et al. [5] proposed to resolve inconsistency based on human choices.

Some works are not directly related to context inconsistency, but address similar problems. Capra et al. [2] used a sealed-bid auction to resolve conflicts among application profiles. Chomicki et al. [3] ignored an inputted event to avoid conflicting actions or randomly discarded some actions to resolve a conflict. Nentwich et al. [9] presented a technique to generate repair options to resolve document inconsistency, and a user should select a right choice. Roman et al. [11] suggested setting up rule priorities to follow human preferences. These works were unsuitable for pervasive computing because they did not support context-awareness in inconsistency resolution.

In this paper, we explain how to respect context-awareness in inconsistency resolution. The paper is complementary to our previous work on inconsistency resolution [13], which requires human participation and does not consider the impact on the context-awareness of applications.

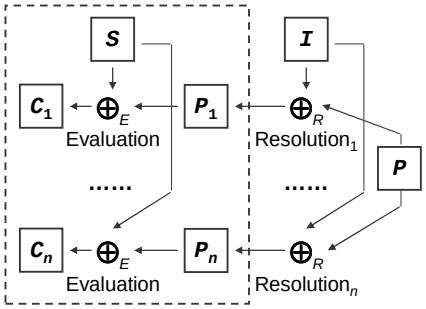

Figure 2. Try all resolutions to find the one that minimizes the impact on the context-awareness of applications

3. IMPACT-ORIENTED RESOLUTION

3.1 Problem Abstraction

We first study two concepts. First, since an environment keeps changing in pervasive computing and valid contexts rarely last forever, only a subset of all contexts which are ever produced is interesting to applications at a point of time. These contexts are referred to as *active contexts*. Second, applications access active contexts by various means, e.g., published/subscribed topics [15], queries/answers [11], and contextual conditions [4]. These means are referred to as *situation specification*, which contains the information about what contexts are needed by applications.

An ideal inconsistency resolution is to make active contexts inconsistency-free for the use by applications. This is illustrated in Figure 1. P represents a set of active contexts, and I represents a set of inconsistencies detected in P (how to detect inconsistencies in contexts is explained in [14]). Inconsistency resolution $\oplus_R$ accepts P and I, and produces P', which represents a set of resolved active contexts (some contexts from P may be discarded in P'). Then, resolved active contexts are evaluated on the situation specification S to decide which of them (C) are needed by applications. This is called *situation evaluation* ($\oplus_E$).

Given P, each resolution strategy may result in a unique P'. Hence, different strategies may bring different impacts on the calculation of C, and thus affect the context-awareness of applications. Our investigation is to find a good resolution that minimizes on some metric of this impact.

3.2 Impact-Oriented Resolution

The drop-latest and drop-all strategies specify two distinct resolutions, i.e., discarding the latest context or all contexts related to inconsistency, respectively. Intuitively, the drop-all strategy often has more severe impacts on applications than the drop-latest strategy (our experiments in Section 4 also confirmed that). However, even if we choose to discard one context to resolve every inconsistency, there are still many possible variants. For instance, the drop-latest strategy is one of these variants, and another strategy could be to discard the earliest context. One may also wish to try all possible resolutions to find the one that brings the least impact on the context-awareness of applications.

The above idea is referred to as *impact-oriented resolution*. Figure 2 illustrates the resolution framework. Suppose that n resolutions

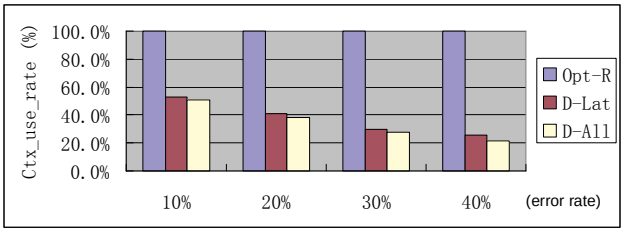

Figure 3. Resolution comparison in Call Forwarding

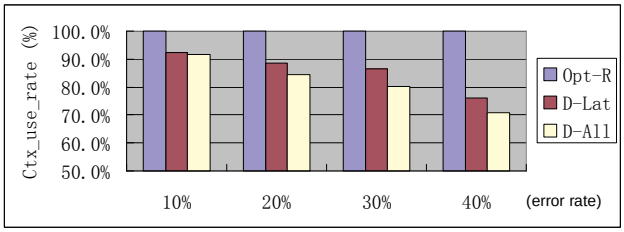

Figure 4. Resolution comparison in RFID data anomalies

are tried. This results in n P's (from P_1 to P_n) and n C's (from C_1 to C_n). The impact-oriented resolution needs to find the resolution $\mathsf{Resolution}_i$ that minimizes $\mathsf{Impact}(C_i)$ among all i's. Here, a metric is needed to measure the impact of inconsistency resolution on context-awareness. Impact is calculated by the difference between the metric value of C_i and a reference value.

3.3 Challenges

In Figure 2, trying every resolution $\mathsf{Resolution}_i$ implies a comprehensive situation evaluation of resolved active contexts P_i with respect to the situation specification S. Trying n resolutions requires n such situation evaluations. Can we alleviate the complexity by heuristics?

3.3.1 Minimizing the Number of Discarded Contexts

One heuristic is to change the goal of minimizing $\mathsf{Impact}(C_i)$ to *minimizing the number of discarded contexts*. Since C_i is no longer needed in the new goal, n situation evaluations are avoidable as they are only used for calculating n C_i for the metric comparison. Is this heuristic useful?

The intuition of this heuristic is that minimizing the number of discarded contexts in inconsistency resolution would guarantee the maximal number of resolved active contexts P'. However, even if we can maximize the number of resolved active contexts P', these contexts may not necessarily be used by applications in their computation. In other words, it is possible that some contexts needed by applications are discarded, and that the remaining contexts after inconsistency resolution are not actually needed.

In addition, even if the discarded contexts are not directly needed by applications, they may help identify other contexts needed by applications. If these contexts are discarded, the other contexts, which can otherwise be used without inconsistency resolution, now become no longer usable for applications.

In the example of Call Forwarding in Section 1, two different resolutions of discarding one context, L_{work} or L_{res}, lead to different results. Although the number of discarded contexts is one for both resolutions, discarding L_{work} makes the application run as if no inconsistency resolution had taken place, whereas discarding L_{res} leads to the result that Call Forwarding does not divert incoming calls to Peter according to the assumed configuration.

3.3.2 Aggregating the Impacts of Individual Contexts

Another heuristic is to *aggregate the impacts of individual contexts to calculate the total impact of a resolution*. In this heuristic, every context is associated with a value, which is the impact on context-awareness when this context is discarded in a resolution. Then, summing up (i.e., aggregating) all the values of the contexts that are discarded in a resolution gives the impact of the whole resolution. Thus, $\mathsf{Impact}(C_i)$ can be calculated without any situation evaluation. This heuristic also avoids n situation evaluations.

Unfortunately, this heuristic is not viable. Let us illustrate this using Call Forwarding in Section 1 to explain the reason. We alter slightly the scenario about the context L_{res}. Suppose that we have two copies of this context, i.e., L_{res1} and L_{res2}. Both of them indicate that Peter is in a restricted zone. Discarding one context L_{res1} or L_{res2} in a resolution brings no impact to the application. This is because either L_{res1} or L_{res2} shows that Peter is in the restricted zone and thus the location contexts of Peter are collected for diverting any call to him. Thus, the associated values of the two contexts are both zero since discarding either of them causes no impact to the application. However, if two contexts L_{res1} and L_{res2} are discarded together in a resolution, the result would be different. Since no remaining context indicates that Peter is in the restricted zone, the location contexts of Peter are missing, and hence no incoming calls may be forwarded to him. This result (non-zero) is not obtainable by aggregation of the impacts of two individual contexts L_{res1} and L_{res2} (both are zero).

4. EXPERIMENTATION

The two heuristics do not work for our problem. This adds to the challenges of building the impact-oriented resolution framework. For the study purpose, we designed simulation experiments to explore how much the context-awareness of applications can be affected by inconsistency resolution using the existing resolution strategies.

4.1 Experiment Design

We conducted the experiments on Cabot middleware [15]. An inconsistency resolution was integrated as a plug-in service, which was invoked when Cabot received new contexts from a simulated client. The inconsistency resolution could be enabled with different resolution strategies under study.

We selected two applications, which were adapted from Call Forwarding [12] and RFID data anomalies [10] (RFID stands for Radio Frequency Identification). We selected five constraints to detect inconsistencies and three situations to use contexts for every application. Contexts were generated with a controlled error rate from 10% to 40%. This is based on the fact that in real-life RFID deployment, the observed read rate (i.e., the percentage of tags in a reader's vicinity that are actually reported) may drop to in the range of 60-70% [6].

In the experiments, we study the existing drop-latest (D-LAT) and drop-all (D-ALL) strategies for inconsistency resolution. For comparison, we assume the existence of an *optimal resolution strategy* (OPT-R). OPT-R has an oracle to discard precisely every erroneous context. Thus, OPT-R represents a theoretical upper bound of a resolution strategy.

The compared measurement was the number of contexts used by applications. All measurement values were normalized to percent-

ages (ctx_use_rate) based on the reference value reported by OPT-R, which was set to 100%. For a resolution strategy, the lower its measurement value is, the more the applications are affected.

4.2 Experimental Results and Analysis

Figure 3 and Figure 4 show the comparison differences, which were averaged over four groups of experiments. In each experiment, 100 contexts with a particular error rate were processed by the middleware equipped with a resolution strategy.

We observe that both D-LAT and D-ALL, as two simple resolution strategies, are overly conservative in inconsistency resolution. In Figure 3, more than 40% of the contexts needed by applications are discarded after inconsistency resolution when the error rate is 10%, and the percentage becomes up to 80% when the error rate is 40%. In Figure 4, the situation is better but the percentage of the discarded context can still be up to 30%. Although the result looks seemingly acceptable (10%) when the error rate is not high (10%), we note that in RFID deployment, the error rate is normally in the range of 30%-40% as explained in Section 4.1. In addition, the result represents how many contexts needed by applications are discarded. Each discarded context may directly affect the context-awareness of the application. Therefore, the result reported in Figure 4 is more serious than the case that the same percentage of active contexts is discarded but not all of them are needed by applications.

Although as expected that D-LAT and D-ALL cannot fully resolve all inconsistencies, the result indicates that they also do not respect the context-awareness of applications satisfactorily.

The results also suggest that an impact-oriented approach is desirable in inconsistency resolution to preserve the context-awareness of applications for pervasive computing. This is because simple resolution strategies that do not respect context-awareness may bring significant impacts to applications as shown in Figure 3 and Figure 4. We observe that a major challenge in building the impact-oriented resolution is to address the computation complexity in evaluating the situation specification (see Section 3.3).

5. CONCLUSION

In this paper, we study the inconsistency resolution problem. We examine the problem from the perspective of pervasive computing and show the need for a special treatment for context-awareness. To reduce the impact of inconsistency resolution on the context-awareness of applications, we propose an impact-oriented resolution framework.

We conducted simulation experiments to show how much the context-awareness of applications may be affected by inconsistency resolution using the existing resolution strategies. The result motivates the need for deploying an impact-oriented resolution for context-aware applications.

To the best of our knowledge, this work is the first attempt at relating inconsistency resolution to context-awareness. Our initial study also reveals some interesting issues, such as the inadequacy of existing resolution strategies for supporting context-aware applications, and the complexity of building an impact-oriented resolution framework. We will continue to conduct further researches in this vein.

ACKNOWLEDGMENTS

This research was supported by a grant from the Research Grants Council of Hong Kong (Project No. 612303).

REFERENCES

[1] Bu, Y., Gu, T., Tao, X., Li, J., Chen, S., and Lv, J. Managing Quality of Context in Pervasive Computing. In *Proceedings of the 6th QSIC*, Beijing, China, Oct 2006, 193-200.

[2] Capra, L., Emmerich, W., and Mascolo, C. CARISMA: Context-Aware Reflective Middleware System for Mobile Applications. *IEEE TSE, 29*, 10 (Oct 2003), 929-945.

[3] Chomicki, J., Lobo, J., and Naqvi, S. Conflict Resolution Using Logic Programming. *IEEE TKDE 15*, 1 (Jan-Feb 2003), 244-249.

[4] Henricksen, K. and Indulska, J. A Software Engineering Framework for Context-Aware Pervasive Computing. In *Proceedings of the 2nd PerCom*, Orlando, USA, Mar 2004, 77-86.

[5] Insuk, P., Lee, D., and Hyun, S.J. A Dynamic Context-Conflict Management Scheme for Group-Aware Ubiquitous Computing Environments. In *Proceedings of the 29th COMPSAC*, Edinburgh, UK, Jul 2005, 359-364.

[6] Jeffery, S.R., Garofalakis, M., and Frankin, M.J.. Adaptive Cleaning for RFID Data Streams. In *Proceedings of the 32nd VLDB*, Seoul, Korea, Sep 2006, 163-174.

[7] Julien, C. and Roman, G.C. EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications. *IEEE TSE 32*, 5 (May 2006), 281-298.

[8] Murphy, A.L., Picco, G.P., and Roman, G.C. LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. *ACM TOSEM 15*, 3 (Jul 2006), 279-328.

[9] Nentwich, C., Emmerich, W., and Finkelstein, A. Consistency Management with Repair Actions. In *Proceedings of the 25th ICSE*, Portland, USA, May 2003, 455-464.

[10] Rao, J., Doraiswamy, S., Thakkar, H., and Colby, L.S. A Deferred Cleansing Method for RFID Data Analytics. In *Proceedings of the 32nd VLDB*, Seoul, Korea, Sep 2006, 175-186.

[11] Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., and Nahrstedt, K. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing 1*, 4 (Oct-Dec 2002), 74-83.

[12] Want, R., Hopper, A., Falcao, V., and Gibbons, J. The Active Badge Location System. *ACM TOIS 10*, 1 (Jan 1992), 91-102.

[13] Xu, C. and Cheung, S.C. Inconsistency Detection and Resolution for Context-Aware Middleware Support. In *Proceedings of the Joint 10th ESEC and 13th ACM SIGSOFT FSE*, Lisbon, Portugal, Sep 2005, 336-345.

[14] Xu, C., Cheung, S.C., and Chan, W.K. Incremental Consistency Checking for Pervasive Context. In *Proceedings of the 28th ICSE*, Shanghai, China, May 2006, 292-301.

[15] Xu, C., Cheung, S.C., Lo, C., Leung, K.C., and Wei, J. Cabot: On the Ontology for the Middleware Support of Context-Aware Pervasive Applications. In *Proceedings of the BISON Workshop*, Wuhan, China, Oct 2004, 568-575.