On Impact-Oriented Automatic Resolution of Pervasive Context Inconsistency[1]

Chang Xu[*] Shing-Chi Cheung[*] Wing-Kwong Chan[#] Chunyang Ye[*]

[*] Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China

[#] Department of Computer Science, City University of Hong Kong, Kowloon Tong, Kowloon, Hong Kong, China

{changxu, scc}@cse.ust.hk, wkchan@cs.cityu.edu.hk, cyye@cse.ust.hk

ABSTRACT

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

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – *Validation.*

General Terms

Measurement, Performance, Verification.

Keywords

Pervasive Computing, Inconsistency Resolution.

1. INTRODUCTION

With the advancement of radio frequency, sensor network and wireless communication technologies, pervasive computing becomes 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 context include object location, time, light intensity and ambient temperature. To ease the use of such 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 matched contexts [11]; contextual conditions are specified and checked to invoke certain application components at right time [4].

Since context-aware applications rely on their acquired contexts to conduct tasks, the quality of these contexts is important to the success of the applications. However, contexts in pervasive computing are naturally error-prone [1]. As a result, contexts are subject to *inconsistency* (i.e., violating predefined consistency constraints). Unexpected outcomes can occur when the applications use inconsistent contexts in conducting their tasks.

Various strategies for resolving inconsistency have been studied. The drop-latest strategy discards the latest context if it causes any inconsistency with existing contexts [3]. The drop-all strategy simply discards all contexts involved in 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 aforementioned drop-latest and drop-all strategies are two examples of automated resolution strategies. They are simple to implement, but overlook potential impact on the context-awareness feature of applications that use contexts.

We illustrate the problem using the Call Forwarding [12] application, which automatically forwards incoming calls to its callees using the nearest phones. The location contexts of all users should be managed carefully to avoid 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 inside a 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 would be inconsistent together. To resolve this inconsistency by discarding contexts, one may consider discarding: (1) L_{work} only, (2) L_{res} only, or (3) both L_{work} and L_{res}.

To study the different impact of the three alternatives on the application, we assume that Call Forwarding is configured to forward a call only when its targeted person is in a mobile phone restricted zone. This imposes a requirement to collect continually the location contexts of a person if there is any context showing that this person enters 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 context L_{res} only (or both L_{work} and L_{res}) would cause no more new location context about Peter to be collected. This is because the only 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 one chooses to discard context L_{work} only, the remaining context L_{res} can still indicate that Peter enters the restricted zone. Then Call Forwarding can continually collect Peter's later location contexts, and forward a call to Peter if any.

When Peter is in the restricted zone, the use of contexts in the first

[1] Crafted version: Made in Feb 2017 (originally published at ESEC/FSE in Sep 2007). Contact: Chang Xu (changxu@nju.edu.cn).

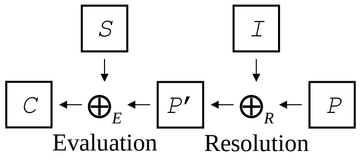

Figure 1. Problem abstraction.

alternative is desirable, because it has the same effect as if no inconsistency resolution had taken place. On the contrary, the context-awareness feature of the application can be adversely affected by the last two alternatives, because incoming calls can no longer be automatically forwarded to Peter without his location contexts being continually collected.

This 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 correct functioning of an application could be unexpectedly affected (e.g., forwarding a call or not) by inconsistency resolution. This perspective has not been explicitly studied in the existing work. We thus formulate two research questions for studying in this paper:

- *Can the impact of an inconsistency resolution strategy on context-awareness be calculated?*
- *Does one have any helpful heuristics for facilitating the calculation of certain 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 in context inconsistency resolution. Section 4 presents our preliminary simulated experiments on two context-aware applications. Finally, Section 5 concludes the paper.

2. RELATED WORK

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

We focus on automated inconsistency resolution in this paper. Some pieces of related work 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 pieces of work did not directly relate to context inconsistency, but addressed 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 users should select a right choice. Roman et al. [11] suggested setting up rule priorities to follow human preferences. These pieces of work were unsuitable for pervasive computing because they did not take context-awareness into consideration 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 of inconsistency resolution on the context-awareness feature of applications.

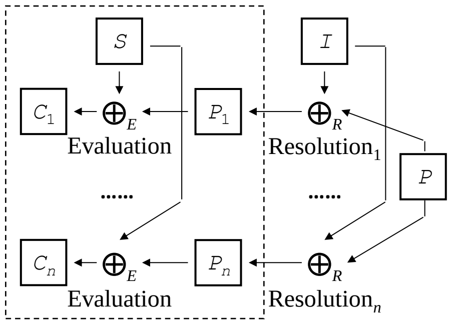

Figure 2. Try all resolutions to select the best one that minimizes the impact on context-awareness.

3. IMPACT-ORIENTED RESOLUTION

In this section, we present the impact oriented inconsistency resolution problem and our approach aiming to respect context-awareness for pervasive computing applications.

3.1 Problem Abstraction

We introduce two concepts. First, since an application's environment keeps changing in pervasive computing and valid contexts rarely last forever, only a subset of all contexts that are ever produced is interesting to applications at each time point. Such contexts are referred to as *active contexts*. Second, pervasive computing 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 specifications*. We assume that a situation specification contains the information about what contexts are needed by applications.

An ideal inconsistency resolution should aim to make active contexts inconsistency-free for use by applications. We illustrate this by Figure 1. P represents a set (pool) of active contexts, and I represents a set of inconsistencies detected in P (how to detect context inconsistency is explained in existing work such as [14]). Inconsistency resolution $\oplus_R$ accepts P and I as input, 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 situation specification S to decide which of them (C) are needed (and thus used) by applications. We name this evaluation step *situation evaluation* ($\oplus_E$).

Given active contexts P, each resolution strategy can result in different resolved active contexts P'. Therefore, different strategies could bring different impact on the calculation of used contexts C, thus affecting the context-awareness feature of applications. Our study is to find a good resolution strategy that minimizes on some selected metric of this impact.

3.2 Impact-Oriented Resolution

The existing drop-latest and drop-all strategies suggested two distinct ways for inconsistency resolution, i.e., discarding the latest context or all contexts involved in any inconsistency, respectively. Intuitively, the drop-all strategy could have more severe impact on applications than the drop-latest strategy. However, even if one chooses to discard only one context to resolve each inconsistency, there are still many possible choices (variants). For example, the drop-latest strategy is one of such variants, and another strategy

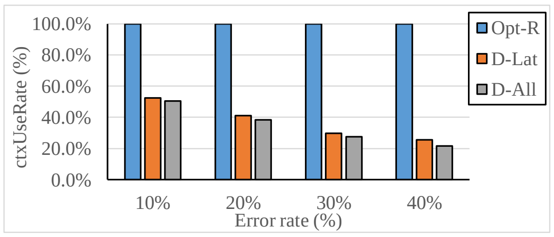

Figure 3. Resolution comparison in Call Forwarding.

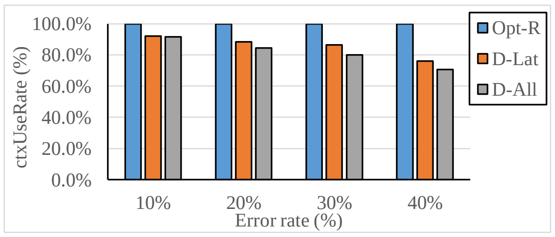

Figure 4. Resolution comparison in RFID data anomalies.

could discard the earliest context, assuming that it has been obsolete. One may also be aggressive and try all possible resolutions to select the best one that brings the least impact on context-awareness of applications.

We refer to this try-and-select idea as *impact-oriented resolution*. Figure 2 illustrates such a resolution framework. Suppose that n resolutions can be tried. This results in n versions of P' (from P_1 to P_n) and n versions of C (from C_1 to C_n). The impact-oriented resolution needs to find Resolution$_i$ that minimizes Impact(C_i) among all n versions. 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$_i$ implies a comprehensive situation evaluation of resolved active contexts P_i with respect to situation specification S. Trying n resolutions would require n such situation evaluations, which could be very time-consuming. Can one alleviate the complexity by heuristics?

3.3.1 Minimizing Number of Discarded Contexts

One heuristic is to change the goal of minimizing Impact(C_i) to *minimizing the number of discarded contexts*. Since C_i is no longer needed in the new goal, n situation evaluations can be avoided since they are only used for calculating n C_i versions for the metric comparison. Then, is this heuristic useful?

The intuition behind this heuristic is that minimizing the number of discarded contexts in inconsistency resolution can guarantee the maximal number of resolved active contexts P'. However, even if one can maximize the number of resolved active contexts, these contexts may not necessarily be used by applications. In other words, it is possible that some key contexts needed by applications are unfortunately discarded in inconsistency resolution, and that the remaining contexts are not so useful for applications.

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

In our earlier discussed Call Forwarding example, two different resolutions of discarding one context, L_{work} or L_{res}, have led to different results, as we analyzed. Although the number of discarded contexts is both one for two resolutions, discarding L_{work} makes the application work as if no inconsistency resolution had taken place (desirable), while discarding L_{res} leads to the result that Call Forwarding no longer forwards incoming calls to Peter (not desirable).

3.3.2 Aggregating Impact of Individual Contexts

Another heuristic is to *aggregate the impact of individual contexts to speed up the calculation of total impact of a resolution*. In this heuristic, each context is associated with a value, which measures the impact on context-awareness when this context is discarded in inconsistency resolution. Then, summing up (i.e., aggregating) all such values of the contexts that are discarded in inconsistency resolution can give the impact of the whole resolution. By doing so, Impact(C_i) can be calculated without any situation evaluation, and thus this heuristic could avoid n comprehensive situation evaluations.

Unfortunately, this heuristic also does not work out. We illustrate this still using the Call Forwarding example. We alter slightly the scenario about context L_{res}. Suppose that one has two copies of this context, i.e., L_{res1} and L_{res2}. Both of them indicate that Peter is in a restricted zone. Discarding only one context, L_{res1} or L_{res2}, in inconsistency resolution would bring no impact to the application. This is because either context L_{res1} or L_{res2} indicates that Peter is in the restricted zone and thus Peter's location contexts would be continually collected for forwarding incoming calls to him. Due to this fact, the associated values for the two contexts would both be set to zero, since discarding either of them can cause no impact to the application. However, if both contexts, L_{res1} and L_{res2}, are unfortunately discarded in inconsistency resolution, the result would be totally different. Since no remaining context indicates that Peter is in the restricted zone, Peter's location contexts would no longer be collected and no incoming call would be forwarded to him. This result (clearly non-zero impact) essentially cannot be obtained by aggregation of impact of two individual contexts L_{res1} and L_{res2} (both carrying zero impact).

4. EXPERIMENTATION

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

4.1 Experimental Design

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

We selected two applications, which were adapted from Call Forwarding [12] and RFID data anomalies [10] (RFID stands for Radio Frequency Identification technology). We selected five consistency constraints for detecting context inconsistencies and three situations for using contexts in each application. Contexts were generated with a controlled error rate from 10% to 40% at a pace of 10%. This is based on the fact that in real-life RFID deployment, the observed read rate (i.e., percentage of tags in a reader's vicinity that are actually detected) 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 purposes, we assume the existence of an artificial *optimal resolution strategy* (OPT-R). OPT-R has an oracle to discard precisely each erroneous context. Thus, OPT-R represents a theoretical upper bound for the best resolution strategy.

The compared metric in the experiments is the number of contexts used by an application. All metric values were normalized to percentages (ctxUseRate) based on the reference value reported by OPT-R, which was set to 100%. For a certain resolution strategy, the lower its metric value is, the more applications are affected in context-awareness.

4.2 Experimental Results and Analysis

Figure 3 and Figure 4 show the comparison results, which were averaged over four groups of experiments. In each experiment, 100 contexts with a controlled error rate were processed by the Cabot middleware enabled with a certain resolution strategy.

We observe that both D-LAT and D-ALL, as two simple resolution strategies, were overly conservative in inconsistency resolution. In Figure 3, more than 40% contexts needed by the application were discarded after inconsistency resolution when the error rate was set to 10%, and this percentage value became close to 80% when the error rate reached 40%. In Figure 4, the situations were better, but the percentage of discarded contexts could still be up to about 30%. Although this result seemed acceptable (about 10%) when the error rate was not high (e.g., 10%), we note that in practical RFID deployment, its error rate would normally fall in the range of 30–40% [6][10]. Besides, the results represent how many contexts needed by applications were discarded. Each discarded context could directly affect context-awareness of applications. Therefore, the results reported in Figure 4 are more serious than the case where active contexts of the same percentage are discarded, among which not all are actually needed by applications.

Although as expected D-LAT and D-ALL could not fully resolve all inconsistencies, the experimental results also indicate that they did not respect context-awareness of applications satisfactorily.

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

5. CONCLUSION

In this paper, we studied the context inconsistency resolution problem. We examined the problem from the perspective of pervasive computing, and analyzed the need for a special treatment for the context-awareness feature of applications. To reduce the impact of inconsistency resolution on context-awareness of applications, we proposed an impact-oriented resolution framework.

We conducted simulated experiments to show how much context-awareness of applications could be affected by inconsistency resolution using existing resolution strategies. The results motivated the need for deploying an impact-oriented resolution for context-aware applications.

To our best knowledge, this work is the first attempt for relating context inconsistency resolution to context-awareness of applications. Our initial study also revealed 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 work along this line.

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*, pp. 193-200, Beijing, China, Oct 2006.

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

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

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

[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*, pp. 359-364, Edinburgh, UK, Jul 2005.

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

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

[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)*, pp. 279-328, Jul 2006.

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

[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*, pp. 175-186, Seoul, Korea, Sep 2006.

[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)*, pp. 74-83, Oct/Dec 2002.

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

[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*, pp. 336-345, Lisbon, Portugal, Sep 2005.

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

[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*, pp. 568-575, Wuhan, China, Oct 2004.