Managing Quality of Context in Pervasive Computing[*]

Yingyi Bu[1], Tao Gu[2], Xianping Tao[1], Jun Li[1], Shaxun Chen[1], Jian Lu[1]
[1]National Laboratory for Novel Software Technology, Nanjing University
Nanjing City, P.R.China, 210093
[2]Institute for Infocomm Research, Singapore, 119613
byy@ics.nju.edu.cn, tgu@i2r.a-star.edu.sg

Abstract

Context-awareness plays a key role in a paradigm shift from traditional desktop styled computing to emerging pervasive computing. Many context-aware systems have been built to achieve the vision of pervasive computing and alleviate the human attention bottleneck; however, these systems are far from real world applications. Quality of context is critical in reducing the gap between existing systems and real-life applications. Aiming to provide the support of quality of context, in this paper, we propose a novel quality model for context information and a context management mechanism for inconsistency resolution. We also build a prototype system to validate our proposed model and mechanism, and to assist the development of context-aware applications. Through our evaluations and case study, context-aware applications can be built with the support of quality of context.

Keywords: context-aware systems, context model, quality of context, pervasive computing.

Topics: emerging technology−pervasive computing, applications, software quality.

Paper type: research paper

1 Introduction

In the recent years, context-aware computing has extracted a lot of attention from academic researchers and industrial practitioners. Context-aware systems usually make use of a large amount of sensed context information which is obtained from various physical sensors. Over the past decade, many context-aware applications have been built; however, few of them has been deployed in real life. One of the critical issue is quality of context; this issue is either ignored or not well addressed in the existing context-aware systems. The issues on quality of context may vary from type to type and from application to application; but it has two main factors. First, inconsistent contexts may often appear in context-aware systems because different sensors may produce different sensed data values which will lead to the inconsistency of sensor-based applications. Second, most sensors usually send sensed data to sinks periodically so that it is very difficult for computers to know what indeed happens in the time interval between two sensor signals. As a result, quality of context is always difficult to guarantee.

In this paper, we propose a management mechanism for quality of context to resolve inconsistency among various context information. This mechanism is based on an ER-ontology based model with the extension of quality measurements. To validate our model and mechanism, we have built a prototype system to enable the development of context-aware applications with the support of quality of context. Through our experiments, we conclude that our approach is feasible in practice, and context-aware applications can be built with the enhancement of quality of context.

The rest of the paper is organized as follows. In section 2, we discuss some related work. Section 3 discusses context quality measurements. Section 4 describes the ER-ontology based context model. Section 5 presents the management mechanism for quality of context. Section 6 introduces our prototype implementation. We evaluate our system in Section 7; and present a case study in Section 8. Finally, we conclude in Section 9.

2 Related Work

In the past decade, many context-aware systems are developed in both research communities and industry companies.

Active Badge [18] is the earliest context-aware applications that redirects phone calls based on people's location. Salber developed Context-Toolkit [17] which is a well designed object-oriented framework supporting context-aware computing. Context Fabric [12] is an infrastructure for

[*]This work is funded by 973 of China (2002CB312002) and 863 Program of China(2005AA113030), NSFC (60233010, 60403014) , NSFJ(BK2006712).

building context-aware applications, which provides a context specification language. ActiveCampus [9] builds a context-aware computing infrastructure by using a central server for all components except data acquisition and user interaction, in order to ease administration and to minimize requirements placed on mobile devices. Context Cube [11] gave a context management mechanism based on the techniques of data warehousing and data mining. The MobiPADS system [5] is a reflective middleware designed to support context-aware processing by providing an execution platform to enable active service deployment and reconfiguration of the service composition in response to environments of varying contexts. CARISMA [4] is also a reflective middleware supporting for mobile context-aware applications, and it can commendably resolve conflict behaviors among different applications. Siren [13] is a real-time context-aware system used in fire fighting domain. The CORTEX [1] project has built a context-aware middleware based on the Sentient Object Model. CoBrA [7] is an agent-based architecture employing ontology based context model for smart room environments. SOCAM [10] builds an OSGi based central service that retrieves context data from distributed context providers, processes them using ontology reasoning and rule reasoning, and offers them to its clients. Solar [6] is a middleware system gathering physical or virtual context information, together with filters, transformers and aggregators modifying context to offer the application usable context information.

Some recent work has been brought in towards issues related to quality of context. Myllymaki proposed a good solution for resolving conflicts in location information [15], but the strategy is difficult to extend for inconsistency detection and resolution of various contexts. Ranganathan tried to resolve semantic contradictiously context using fuzzy logic in first predicate caculus [16]. Dey gave a novel solution for ambiguity resolution by user mediation [8]. Xu established a context consistency management mechanism by providing a sophisticated architecture for inconsistency detection and resolution [19], and using an well-designed incremental consistency checking approach [20]. Our inconsistency resolution is performed at raw context level without human interference. In our previous work [2][3], time constraints are attached to high-level contexts and a lifecycle management strategy is brought in. Although quality of context is improved to some extent, applications based on our previous infrastructure still have some uncertainty and often conduct perplexing behaviors because the time constraints of contexts are approximate values. Nevertheless, we find that using ER-ontology based model can largely reduce the inconsistency levels of context information and improve quality of context because updating context repository is content-based.

3 Context Quality Measurements

To measure quality of context, we propose three important parameters: Delay time, Context correctness probability, and Context consistency probability.

Delay Time. Delay time is the time interval between the time when the situation happens in real world and the time when the situation is recognized in computers. It is important to context-aware applications, because outdated contexts will not be useful to applications.

Context Correctness Probability. Due to the limitation of sensor technology, the accuracy of sensed data is difficult to guarantee. However, if we measure contexts through random sampling in a rather long period with recording the correct rate(the probability that contexts in computers match situations in real world), we are able to provide quantity measurement for quality of context in a context-aware system.

Context Consistency Probability. Incorrect contexts often lead to context inconsistency. Context Consistency Probability measures the consistency rate of context information(the probability that contexts in computers are consistent), and it also could be obtained through long period random sampling.

A well-designed context-aware system should have low delay time, high context correctness probability and high context consistency probability. The three measurements have correlations with each other: outdated contexts with large delay time are usually incorrect and conflicting with current context information; and inconsistent contexts usually contain incorrect ones.

In the following sections, we intend to improve context quality by dealing with the three measurements respectively. We adopt an ER ontology model as our basis, and propose the context inconsistency resolution algorithm, the raw level refactoring, the particular callback mechanism to promote context correctness probability and context consistency probability; and replace high-level inconsistency resolution with raw-level inconsistency resolution to effectually shorten delay time.

4 ER-ontology Based Context Model

4.1 Conceptual Model−EROntCom

The EROntCom consists of ontology, entities, relationships, and dependencies.

The ontology is a set of shared vocabularies of concepts and the interrelationships among these concepts. With the ontology, ontology-based reasoning and rule-based reasoning can be used to infer implicit contexts. Contexts can be easily shared in different computational nodes, and inconsistency need not be defined explicitly by developers.

Every entity in the physical environment is modeled as “entity” in computer’s view, such as person, desk, classroom, meetingroom, and campus.

Relationships denote the situations or predicates about entities. For example, "Tom locates in Room311" is a context, in which Tom is an entity of type "Person", Room311 is an entity of type "Room", and "locateIn" is the relationship between Tom and Room311. There are two types of relationships, "directRelationship" and "implictRelationship". A case in point, when Prof. Jimmy seats on the chair in his office Room312, the system will sense a context $context_a$ "Jimmy locateIn Room312" directly from some location sensors. We say "locateIn" in $context_a$ is a "directRelationship". The system then can infer an implicit context $context_b$ "Jimmy locateIn MMWBuilding" because Room312 locates in MMWBuilding and the "locateIn" relationship is transitive. We define "locateIn" in $context_b$ as "implicitRelationship". Furthermore, each raw relationship also has some description data, such as frequency which indicates refresh times in a period, starttime which denotes when the relationship is established, updatetime which denotes when it is updated most recently and ttl which describes the maximum updating interval. Then, we can define two categories of relationships in raw context layer: single-value relationship and multi-value relationship. For example, in context "Tom locates in Room311", "locateIn" is a single-value relationship because in raw context layer, Tom can only be in one physical space. In another example, the relationship "talkTo" in contexts "Tom talk to Jimmy" and "Tom talk to Bob" is a multi-value relationship because Tom can talk to more than one person at the same time.

In each high-level context, the "relationship" element is "implicitRelationship" so that it depends on its derivation contexts. "Dependency" is the relationship between "Relationship"s. For example, "locateIn" in $context_b$ depends on the "locateIn" in $context_a$ so there is a dependency between the two "Relationship"s, which starts from "locateIn" in $context_a$, and points to "locateIn" in $context_b$. It is obvious that "Dependency" has a transitivity property.

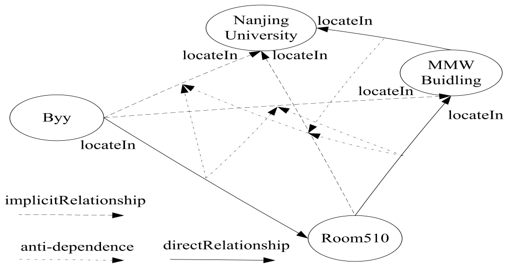
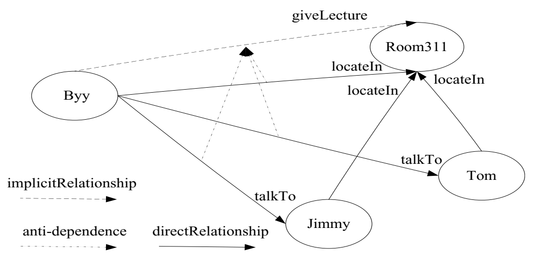


Fig. 1 and Fig. 2 show two examples of context graph. In Fig. 1, the high-level contexts are inferred using ontology-based reasoning while in Fig. 2, the high-level contexts are inferred using rule-based reasoning.



4.2 Formulated Context Model

Context representation and management can be formulated using a graph model. We first describe the following definitions.

Definition 1. Context Graph, A context graph consists of the following elements:

−Node denotes entity in the EROntCom,

−Edge$_{AB}$ denotes the relationship which connects two nodes A and B from A to B.

−Raw-edge denotes the context predicate sensed directly from sensors. Each raw-edge contains four attributes: "ttl" denotes the lifetime of a context; "starttime" is the UNIX time when a context is created in the system while "updatetime" denotes the UNIX time when the context is lately updated; "frequency" indicates how many times a context is updated from its first appearance.

−Implicit-edge denotes the relationship is generated by inference.

−Single-value-edge denotes a single-value relationship.

−Multi-value-edge denotes a multi-value relationship.

−Meta-edge$_{CD}$ is a direct edge between two edges C and D, and the terminal edge D is dependent on the starting edge C.

−Persistent-edge is an edge that always holds, such as "Jimmy is a teacher".

−Dynamic-edge is an edge that holds only for a short period, such as "Jimmy is talking with John".

Definition 2. Context

−Context is a subgraph of context graph, for example, a triple of ($node_A$, $edge_{AB}$, $node_B$), in which $node_A$ is a subject node, $node_B$ is a object node.

Definition 3. ER Graph

−ER graph is a graph which consists of nodes and edges, it is also a subgraph of context graph.

Definition 4. Dependency Graph

−Dependency graph, is a subgraph of context graph which only has edges and meta-edges.

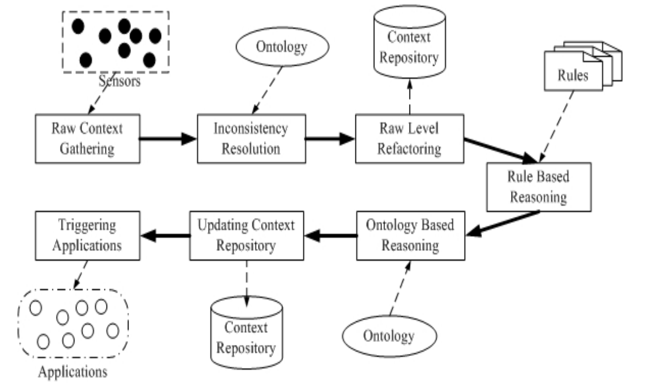
Definition 5. Context Repository

−Context repository, consists of several context graphs.

5 Context Management Mechanism

5.1 Context Processing Procedure

We use logic inference to process contexts in our system. The detailed context processing procedure is shown in Fig. 3. The first step is the raw context gathering, in which raw contexts from various sensor sources are collected during a fixed short period. The second step is the inconsistency resolution. We resolve inconsistency among different raw contexts in this step because inconsistent raw contexts may lead to high-level inconsistent contexts that are more difficult to handle. We process raw contexts in a batch by batch manner instead of a piece by piece manner. Inconsistency in a batch of raw contexts should be cleaned prior to context reasoning so that the inconsistency of high-level contexts can be mitigated in certain degree. The third step is the raw level refactoring, in which we update the context repository with raw contexts, check the dependency graphs and refactor the ER graphs. Outdated or incorrect high-level contexts will be deleted in this step. If they are not removed, they will result in serious inconsistency among contexts after reasoning. Then, we apply rule-based reasoning and ontology-based reasoning based on the Jena API[1] to generate high-level contexts. The user-defined rules are in the form of Jena generic rules without negation and "or" operation. The two reasoners are configured as "traceable" in order to facilitate updating dependency graphs in context repository, though more memory is required. After that, we use inferred high-level contexts to update the context repository and notify applications which register context triggers.



5.2 Inconsistency Resolution in Raw Context Level

The first step of inconsistency resolution is to detect conflicts. For example, if there are 2 raw contexts: d_1(Tom, walkIn, Room311, 15s,1116943120489, 1116943567511, 10) and d_2(Tom, walkIn, Aisle3, 25s, 1116943111897, 1116943567599, 1), 2 persistent contexts: p_1(Room311, type, Room) and p_2(Aisle3, type, Aisle), and 2 assertions in ontology: o_1(Room, disjointWith, Aisle) and o_2(walkIn, type, FunctionalProperty), a conflict will be detected in ontology model because there is an instance of both Room and Aisle.

[1]Jena Semantic Web Toolkit: http://www.hpl.hp.com/semweb/jena2.htm

We first denote some definitions, and then introduce the algorithm using these definitions.

Definition 6. Conflict Pair Set

−Conflict pair set, is a set consisting of pairs such as ($edge_a$,$edge_b$), in which corresponding $context_a$(contains $edge_a$) conflicts with $context_b$(contains $edge_b$). For convenience, we can also say $edge_a$ conflicts with $edge_b$.

Definition 7. Conflict Set

−For a given edge set denoted as $EdgeSet$, if its members conflict with each other, we call $EdgeSet$ a conflict set.

Definition 8. Relative Frequency$-rf$

−A formula that calculates the rf value of a raw-edge $edge_i$ is shown as follow.

$$edge_i.rf = \begin{cases} \dfrac{edge_i.ttl \cdot edge_i.frequency}{currenttime - edge_i.starttime} \\ \text{(for dynamic-edges)} \\ \\ infinite \\ \text{(for persistent-edges)} \end{cases}$$

Our design principle is that more frequent raw contexts have more priority than infrequent ones. However, it is difficult to compare the frequencies for different types of contexts. For example, noise contexts may be inherently varied more frequently than temperature contexts, but we can't say that noise contexts have more priorities. For this reason, we use the rf value to scale each raw context's relative frequency because the ttl value may often imply the context is inherently frequent or infrequent. We believe that those contexts with larger rf values are more frequently recently, therefore, they are more likely to be correct contexts. Hence, when conflict occurs, we discard the ones with smaller rf values. The RCIR(Raw Context Inconsistency Resolution) algorithm is shown in Algorithm. 1. The function $ConflictDetection$ utilizes Jena's consistency checking API. A greedy strategy is used in the algorithm, in which we start from the first element in the conflict pair sets, and form maximum conflict sets circularly to resolve conflicts. The worst case time complexity of RCIR is polynomial bounded with the number of total contexts, and in our experiments we found that it is more efficient than our previous approaches.

5.3 Raw Level Refactoring in Context Repository

Before context reasoning, the context repository should be updated in order to ensure incorrect or outdated high-level contexts are removed. We design an algorithm called

```
Input: Raw Context Set $S_{rc}$
Output: Consistent Set $S_{con}$
1  Conflict Pair Sets $C_{ps}$ = ConflictDetection($S_{rc}$);
2  for every $c_k$ in $C_{ps}$ do
3      add $c_k.edge_1$, $c_k.edge_2$ to $S_{con}$;
4  end
5  while $C_{ps}$ not empty do
6      Conflict Pair Set $c_i$ = first element of $C_{ps}$;
7      Edge $edge_k$=$c_i.edge_1$;
8      Conflict Set $conflict$ = new Conflict Set;
9      for every $c_j$ in $C_{ps}$ do
10         if $c_j.edge_1 == edge_k$ or
           $c_j.edge_2 == edge_k$ then
11             add $c_j.edge_1$, $c_j.edge_2$ to $conflict$;
12             delete $c_j$;
13         end
14     end
15     select a edge $edge_{max}$ with largest $rf$ in
       $conflict$;
16     for every $edge_j$ in $conflicts$ ($j \neq max$) do
17         delete $edge_j$ in $S_{con}$;
18     end
19 end
```

Algorithm 1: Raw Context Inconsistency Resolution(RCIR)

RLR(Raw Level Refactoring, shown in Algorithm. 2) to complete this task.

The algorithm first finds the node in an ER graph which equals to the subject node of a raw context using a DFS[2] search strategy, and then updates the edges and corresponding nodes. For example, if Tom walks from Room311 into Room312, the sensors in Room312 detect Tom within its range, the algorithm will update the edge of Tom's "locateIn" from "Room311" to "Room312" because "locateIn" is a single-value edge in the raw layer. If the edge is multi-valued, the original relation edge will be reserved.

If a raw-edge is changed, the implicit-edges should also be changed. The algorithm will remove edges dependent to the inexistent raw-edges. After dependency checking, if the depending edges is removed, the corresponding dependency will also be deleted. The benefit of our graph model is that when we want to find a node in the context repository, the dependencies are not considered and the algorithm just searches the ER graph; and when we check the dependencies, the algorithm will only operate on the dependency graph, and treat edges as nodes and dependencies as edges without considering entities. The algorithm can be easily extended to handle the input of a raw context array by a simple circulation.

[2]DFS:Depth-first search.

```
Input: Raw Context $rc$, Context Repository
       $repository$
Output: Context Repository $repository$
1  Node $subject$ =
   DFSSearch($repsitory.ERGraphs$, $rc.subject$);
2  for every $edge_i$ out from $subject$ do
3      if $edge_i.type == singlevalue$ and
       $edge_i.name == rc.edge.name$ then
4          Tree $tree$ = DFSTree($edge_i$,
           $edge_i.DependencyGraph$);
5          for every Meta-edge $medge$ in $tree$ do
6              delete $meta - edge$;
7          end
8          for every Edge $edge$ in $tree$ do
9              delete $edge$'s all in-meta-edges and
               $edge$;
10         end
11         delete $edge_i$;
12     end
13 end
14 establish $rc.edge$ from $rc.subject$ to $rc.object$ in
   the ERGraph;
```

Algorithm 2: Raw Level Refactoring(RLR)

5.4 Context Reasoning and Context Repository Updating

We use Jena semantic web API to perform reasoning on the refactored context repository. We translate the ER graph list to RDF[3] triples as inputs to the reasoners and record the reasoning traces when high-level contexts are inferred. After reasoning, the new generated RDF triples are translated back to the ER contexts, and added to ER graphs in the context repository. Finally, the reasoning traces are used to add dependencies to context graphs. Besides, a time tick thread is running periodically to discard outdated raw contexts and refactor context repository using an algorithm similar to RLR.

6 The Software Infrastructure

6.1 System Overview

We have implemented the context management mechanism. We built a centralized OSGi[4] based context service to accept raw contexts from raw context providers, produce high-level contexts, and deliver contexts to applications. In our prototype system, sensors gather data from the physical world, such as location, temperature, pressure, picture, and noise; context providers interact with sensors, and transform sensor data into raw semantic contexts; and applications (context consumers) consume contexts by either subscribing or querying with adaptation to context changes. To assure application's efficiency and robustness, we also design a friendly context query interface and specialized call-

[3]RDF reference: http://www.w3.org/TR/rdf-ref
[4]OSGi, Open Service Gateway Initiative: http://www.osgi.org

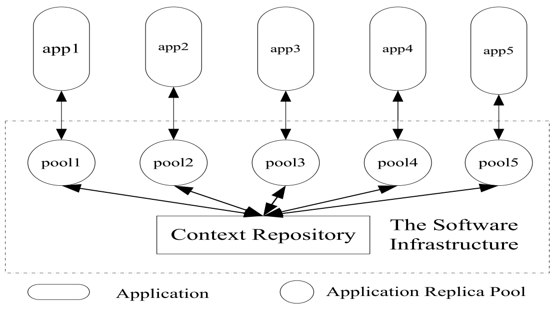
back device.

6.2 Context Query

In our prototype system, we use RDQL[5] as context query language. Applications can query contexts by specifying a RDQL sentence. For instance, we can use “select ?x where (?x giveLecture Room311)(?x Type Teacher)” to search if there is a teacher giving a lecture in Room311. Also, we can look up historical contexts by attaching time ranges to a RDQL sentence. The query mechanism is efficiently implemented on the formulated context model.

6.3 Context Callback

Applications can exploit contexts not only by querying but also by registering callbacks. However, context callback mechanism should be different from conventional event-callback mechanisms. Contexts are varied with time and callbacks must exactly match to real world’s requirements. For example, if a context-aware application intends to open slides for lecturers automatically, with a badly designed callback mechanism, the application may open the slides more than once resulting in confusing users. To address this issue, we invokes callbacks after every inferences and time tick, and use a replica pool to store consumed contexts for each applications. When the callback is being invoked, the system checks each replica pool and does not trigger those stored consumed contexts’ callback functions. Unless those consumed contexts have some changes, they will not be cleared from the replica pool. This device embraces the particularity of contexts and leads to robust applications in practice. The view of callback architecture is shown in Fig. 4.



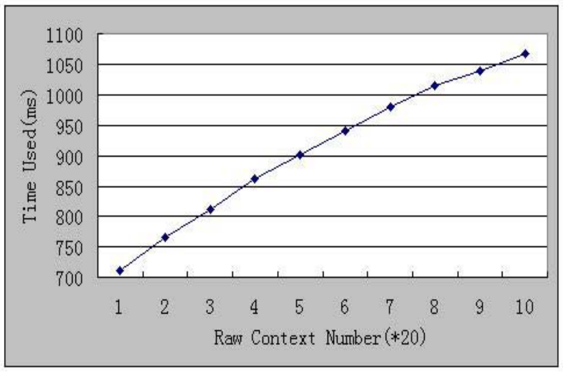
7 Evaluations

We conduct several experiments over our prototype system to evaluate its performance. In our evaluation, the rules are in the form of Jena Generic Rules, and the ontology reasoner is entailed by OWL-Lite. The ontology contains 1257 RDF triples. We use three computers in our experiments, one Linux workstation with 4G RAM and 2 Xeon CPUs and two PC clients, connecting through LAN. In our testbed, the meeting room and aisle are equipped with mica sensors[6] to detect noise and cricket sensors[7] to find people’s location. One of the clients plays the role of a raw context provider while the other acts as a context consumer.

The raw context gathering period mentioned in 4.1 is important to both performance and effectiveness. If the fixed interval is too short, the processing mechanism will retrograde to piece by piece processing which will leads to context conflicts. However, if the interval is too long, the RCIR algorithm will have low performance because it has to to deal with to much inconsistency, and the accuracy will decline because some raw contexts may be outdated.

7.1 Performance Study

In this experiment, we test the performance of the RCIR algorithm on a Linux Workstation with 4G RAM and 2 Xeon CPUs. In the experiment setup, eight persons carrying a cricket sensor and a mica sensor each walk along the aisle and to the meeting room. We investigate the performance with different raw context numbers(edge number), by tuning the raw context gathering period. The results are shown in Fig 5.



7.2 Effectiveness Study

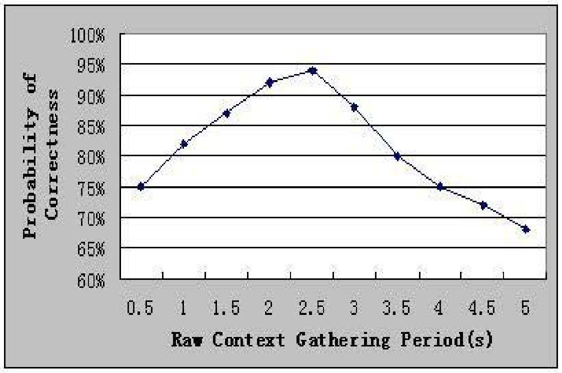
To evaluate the effectiveness of RCIR, in this experiment, a person carrying a cricket beacon stands in a meeting room to give a lecture. During this period, he/she goes out to the aisle with immediately coming back to the meeting room at a frequency of 10 seconds once. This activity can cause context conflicts among high-level contexts in the system because of two raw context triples: $(person_x$, locateIn, $MeetingRoom_x)$ and $(person_x$, locateIn, $Aisle_x)$. We test the effectiveness of RCIR with different raw context gathering interval(horizontal axis in Fig. 6). Meanwhile, for

[5]RDQL tutorial: http://jena.sourceforge.net/tutorial/RDQL/index.html

[6]The Mica Sensor: http://www.xbow.com

[7]The Cricket indoor location system: http://cricket.csail.mit.edu/

each raw context gathering interval, the context consumer continues querying contexts for 10 minutes at a frequency of 10 times per minute to see the probability of context correctness(vertical axis in Fig. 6). In this way, for every raw context gathering interval, 100 samples about context quality can be gained. The results are shown in Fig. 6. We can conclude that 2.5s is an optimal interval.



7.3 Remarks From Experiments

From the experiments, we conclude that RCIR is a very efficient. It improves our previous work [2][3] significantly so that we can run it in normal context processing procedure rather than periodically. However, since there is a fixed number(1257) of RDF triples transformed from the ontology, the consistency checking without raw contexts needs 700ms. This is the limitation of using ontology based consistency checking. The context management mechanism can guarantee high accuracy of contexts if the raw context gathering period is appropriate.

8 Application Case Study

8.1 Scenario

In an academic environment, seminars are often held. When someone gives a lecture, he/she should copy the slides to his/her flash disk, carry it to a meeting room, copy the slides to the computer in the meeting room, and then open them. The work is dull and trivial, and much of people's attention could be distracted. In our context-aware computing environment, the lecturer does not require to all of these. When he/she enters the meeting room, his/her slides will be opened automatically. During the seminar, if some strangers come in, a warning sign will pop up on the screen. At the end of the seminar, the slides will be closed automatically.

8.2 Implementations

We implement three versions of the above scenario: the first one is based on our earlier work which employs a simplistic inconsistency resolution strategy that later updated and persistent contexts are prior(with SIR) [2], the second is based on our previous work in which inconsistency is resolved at high-level by a heuristic algorithm(with CIR) [3], and the third one is based on our context management mechanism(with RCIR).

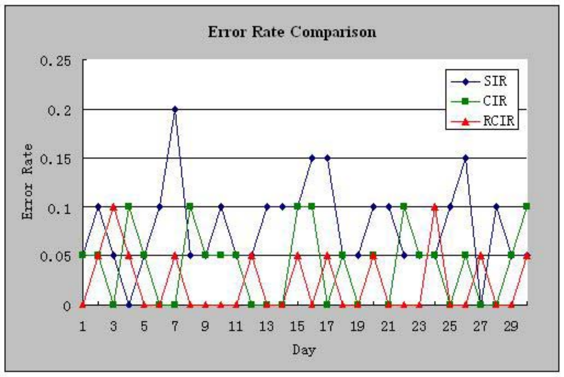
The application called Seminar Assistant has two parts. One called User Assistant runs at all users' computers while the other called Meeting Assistant runs at the computer in the meeting room. When the User Assistant detects the context that the user will give a lecture in the next few days, it will upload the slides he edited, the name of which matches the lecture to an http server. When the lecturer starts to give the lecture in the meeting room, the Meeting Assistant will obtain the right context, and then download and open the previous uploaded slides. Then the Meeting Assistant starts to monitor if strangers come in(a warning sign will pop up if there are). When the Meeting Assistant detects the context that the lecturer leaves the room, it will close the slides.

8.3 Application Error Rate Comparison

We compare the three versions of "Seminar Assistant" by investigating their average error rates. For the comparison, we run the three applications respectively for 600 times(20 times each day, 30 days in total), and record the error rates for each day. In the experiments, all the errors recorded are application's incongruous behaviors such as opening the slides before the reporter entering the meeting room, and system failures such as out of memory error are not included. Fig. 7 shows the results, in which the horizontal axis denotes the day while the vertical axis denotes the error rate. It can be concluded from the experiment results that our context consistency management mechanism(with RCIR) has significantly improved context-aware applications' robustness since many incongruous behaviors are reduced(13 errors of 600 tests, compared to 43 of 600 and 24 errors of 600).

9 Conclusions and Future Work

From our experiences on designing and implementing context-aware applications, we find that quality of context is a critical problem which can threaten the robustness of context-aware applications. This paper proposes several context quality measurements for evaluating context-aware systems. We design an inconsistency resolution algorithm, establish a context management mechanism, and build a software infrastructure to support context-aware applications. Through the evaluations and case study, we prove the feasibility and effectiveness of our mechanism. The work in this paper is part of our ongoing research project−Artemis-FollowMe [14] which is designed towards a workflow-



driven, service-oriented, pluggable and programmable software infrastructure for context-awareness.

In our future work, we will work towards a better inconsistency resolution approach which will further improve quality of context. Further more, we plan to establish a theoretical context quality model which will show to gain a certain context quality, what is the lower bound of sensor energy and computational resource consumptions.

References

[1] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), 14-17 March 2004, Orlando, FL, USA*, pages 361–365. IEEE Computer Society, 2004.

[2] Y. Bu, J. Li, S. Chen, X. Tao, and J. Lu. An enhanced ontology based context model and fusion mechanism. In *Proceedings of IFIP 2005 International Conference on Embedded and Ubiquitous Computing (EUC2005). Nagasaki, Japan.*, volume 3824 of *LNCS*, pages 920–929. Springer, 2005.

[3] Y. Bu, J. Li, S. Chen, X. Tao, and J. Lu. Context consistency management using ontology based model. In *Proceedings of the 2nd International Workshop on Pervasive Information Management (PIM2006), in conjunction with the 2006 International Conference on Extending Database Technology (EDBT2006). Munich, Germany*, pages 21–32, March 2006.

[4] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, 2003.

[5] A. T. Chan and S.-N. Chuang. Mobipads: A reflective middleware for context-aware mobile computing. *IEEE Transactions on Software Engineering*, 29:1072–1085, December 2003.

[6] G. Chen and D. Kotz. Policy-driven data dissemination for context-aware applications. In *Proceedings of 3rd IEEE International Conference on Pervasive Computing and Communications(PerCom2005), Kauai Island, HI, USA*, pages 283–289, 2005.

[7] H. Chen, T. W. Finin, A. Joshi, and L. K. F. Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, pages 69–79, November 2004.

[8] A. K. Dey and J. Mankoff. Designing mediation for context-aware applications. *ACM Transactions on Computer-Human Interaction(TOCHI)*, 12(1):53–80, 2005.

[9] W. G. Griswold, R. S. Boyer, S. W. Brown, and T. M. Truong. A component architecture for an extensible, highly integrated context-aware computing infrastructure. In *Proceedings of the 25th International Conference on Software Engineering (ICSE2003), Portland, Oregon, USA*, pages 363–373. IEEE Computer Society, 2003.

[10] T. Gu, H. K. Pung, and D. Q. Zhang. Towards an osgi-based infrastructure for context-aware applications in smart homes. *IEEE Pervasive Computing*, pages 66–74, December 2004.

[11] L. D. Harvel, L. Liu, G. D. Abowd, Y.-X. Lim, C. Scheibe, and C. Chatham. Context cube: Flexible and effective manipulation of sensed context data. In *Proceedings of the Second International Conference on Pervasive Computing(PERVASIVE 2004), Vienna, Austria*, volume 3001 of *LNCS*, pages 51–68. Springer, 2004.

[12] J. I. Hong and J. Landa. An infrastructure approach to context-aware computing. *Human-Computer Interaction (HCI) Journal*, 16, 2001.

[13] X. Jiang, N. Y. Chen, J. I. Hong, K. Wang, L. Takayama, and J. A. Landay. Siren: Context-aware computing for firefighting. In *Proceedings of The Second International Conference on Pervasive Computing(PERVASIVE2004), Vienna, Austria*, pages 87–105, 2004.

[14] J. Li, Y. Bu, S. Chen, X. Tao, and J. Lu. Followme: On research of pluggable infrastructure for context-awareness. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA2006)*, volume 1, pages 199–204. IEEE Computer Society, 2006.

[15] J. Myllymaki and S. Edlund. Location aggregation from multiple sources. In *Proceedings of the Third International Conference on Mobile Data Management (MDM 2002), Singapore, January 8-11, 2002*, pages 131–138. IEEE Computer Society, 2002.

[16] A. Ranganathan, J. Al-Muhtadi, and R. H. Campbell. Reasoning about uncertain contexts in pervasive computing environments. *IEEE Pervasive Computing*, 03(2):62–70, 2004.

[17] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceeding of the CHI 99 Conference on Human Factors in Computing Systems: The CHI is the Limit (CHI99), Pittsburgh, PA, USA. ACM, 1999*, pages 434–441, 1999.

[18] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.

[19] C. Xu and S.-C. Cheung. Inconsistency detection and resolution for context-aware middleware support. In *Proceedings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2005), Lisbon, Portugal*, pages 336–345, September 5-9 2005.

[20] C. Xu, S.-C. Cheung, and W. K. Chan. Incremental consistency checking for pervasive context. In *28th International Conference on Software Engineering(ICSE 2006), Shanghai, China, May 20-28, 2006*, pages 292–301, 2006.