Context Consistency Management Using Ontology Based Model *

Yingyi Bu, Shaxun Chen, Jun Li, Xianping Tao, and Jian Lu

National Laboratory for Novel Software Technology, Nanjing University
Nanjing City, P.R.China, 210093
byy@ics.nju.edu.cn,buyingyi@nju.org.cn

Abstract. Inconsistent contexts are death-wounds which usually result in context-aware applications' incongruous behaviors and users' perplexed feelings, therefore the benefits of context-aware computing will become less believed. This problem occurs in most sensor based applications due to the intrinsic drawbacks of fallible physical sensors which can only detect some evidence of real world's situations rather than global views of them. In this paper, we extend ontology based context modeling approach with some descriptive information added to contexts, modify reasoners to support time information, bring in a context life-cycle management strategy, establish a context exploitation mechanism, and propose an inconsistency resolution algorithm, fostering timely, exact and conflict-free contexts. Besides, evaluations and a case study are carried out to attest our design principles.

1 Introduction

Context-awareness which aims at decreasing people's attentions to various computational devices is an attractive feature of pervasive computing paradigms. Context informs both recognition and mapping by providing a structured, unified view of the world in which the system operates [1]. However, context is different from knowledge in traditional views because of its dynamic, transient, and fallible characteristics.

It is widely acknowledged that a good context model can lead to well designed and easily understood context-aware applications. Recently emerging ontology based context modeling approach [2][3][4] is an elegant solution towards context sharing, reasoning and reusing. However, in practice, context-aware applications are so fragile that their behaviors often make users bewildered, due to mismatching between contexts in computer systems and contexts in real world. Concretely speaking, inconsistent contexts often appear in context-aware systems on account of failures from either physical sensors or software infrastructures. For example, contexts like "Tom is giving a lecture" and "Tom is talking to Jim on the Aisle" may appear at the same time. How do we know which context is correct? Our previous work [5][6] focuses on context model and fusion infrastructure design, but context management and conflict resolution are simply considered so that applications based on the infrastructure are not so robust.

* This work is funded by NSFC (60233010, 60273034, 60403014), 973 Program of China (2002CB312002), 863 Program of China (2005AA113160) and NSF of Jiangsu Province (BK2002203, BK2002409).

Nevertheless, we find ontology based context model can largely facilitate inconsistency detection and resolution. In this work, we first extend ontology based context model by adding some descriptive information such as time, frequency and state to contexts. Based on the context model, a context management device is established, which not only aims at timely and accurate contexts but also facilitates inconsistency resolution. Then, we design an inconsistency resolution algorithm to provide correct and consistent contexts. Through experiments and an application case study, we find that our modified approach is rather acceptable for fetching up those disadvantages in previous work and the quality of contexts is largely improved.

The rest of this paper is organized as follows. In Section 2, we discuss some related work. The extended context model is presented in Section 3. Our context management mechanism is proposed in Section 4. Section 5 introduces our context inconsistency resolution algorithm CIR. The evaluations are given in Section 6. Section 7 presents a case study to verify our design principles. Finally, we conclude in Section 8.

2 Related Work

In the past decade, many context-aware systems are developed both in research communities and industrial companies which all contribute a lot to context-aware computing.

Active Badge [7] is the earliest context-aware applications that redirects phone calls based on people's locations. Salber developed Context-Toolkit [8] which is a well designed object-oriented framework supporting context-aware computing. Context Fabric [9] is an infrastructure for building context-aware applications, which provides a context specification language. Solar [10] is a middleware system that consists of various information sources such as sensors, gathering physical or virtual context information, together with filters, transformers and aggregators modifying context to offer the application usable context information. CoBrA [11] is an agent-based architecture employing ontology based context model for smart room environments. SOCAM [4][3] proposed an ontology based context model addressing context sharing, reasoning and knowledge reusing, and built a service oriented middleware infrastructure for applications in a smart home. Cooltown [12] is a web based context-aware system. The CORTEX [13] project has built a context-aware middleware based on the Sentient Object Model, in which there is an event-based communication mechanism supporting loose coupling between sensors, actuators and application components. CASS [14] enables developers to overcome the memory and processor constraints of small mobile computer platforms with supporting a large number of low-level sensor and other context inputs, and opens the way for context-aware applications configurable by users. Context Cube [15] gives a good context management mechanism based on the techniques of data warehousing and data mining. Siren [16] is a good real-time context-aware system used in fire fighting domain. Sparkle [17] is a flexible platform to support context-aware services with migrations on difference type of devices.

In previous systems and researches, many context modeling approaches are proposed, either formal or informal, including key-value, object, XML, ER-UML, ontology and so on [18]. Ontology based context model and reasoning mechanism proposed in [2][4][3] displays its potential value for most non-time-critical applications. Kalyan

[19] presented a hybrid context model based on multilevel situation theory and ontology to handle complex user's queries by creating simple entity specific situations and enable efficient context reasoning. Strimpakou [20] built a well designed context management architecture in distributed environments.

But none of the works above concern context conflict resolution in one computational node nor introduce their conflict resolution algorithm. Dey [21] gave a novel solution for ambiguity resolution by user mediation, while Xu established a context consistency management mechanism by providing a sophisticated architecture for inconsistency detection and resolution [22], and using a well-designed incremental consistency checking approach [23]. But differently, our intention is to resolve context ambiguity automatically in software infrastructure layer. Although Myllymaki [24] proposed a good solution for resolving conflicts in location information, the strategy is difficult to be extended for inconsistency detection and resolution of various contexts.

In addition, our modifications of ontology based context model is totally different from temporal databases [25] because we deal with time constraints during context fusions upon ontology based model, and those time constraints are used in inconsistency resolution .

3 pvCM—The Extended Context Model

3.1 Conceptual Model

Our modeling approach is still ontology based, but for convenient context management and inconsistency resolution, some extensions are brought in. The extended context model called pvCM consists of 2 parts: ontology and its instances(including both persistent contexts and dynamic contexts). The ontology is a set of shared vocabularies of concepts and the interrelationships among these concepts. Persistent contexts are instances of the ontology and they can be combined with dynamic contexts during inferences. Triples described as (subject, predicate, object) are used to model persistent contexts which can last a long period. For example, the context "Tim is a student" is modeled as (Tim, type, Student). Dynamic contexts with transient characteristics only have a short life in the system, such as "Jimmy in NJU". Octuples (subject, predicate, object, ttl, starttime, updatetime, frequency, state) are used to represent them. Ttl means the life period of the context. "Starttime" is the UNIX time when the context begins existing in the system while "updatetime" denotes the UNIX time when the context is lately updated. A more important element of dynamic contexts is the "frequency" value which indicates how many times the context is updated from its first appearance. The "state" value describes contexts' life status: "Beginning", "Updated", "Inert", or "Disappearing", the details of which will be explained in Section 4.

3.2 Implementations

The ontology of pvCM is constructed by OWL-Lite[1]. Fig. 1 shows part of our ontology for laboratory office domain. Persistent contexts are serialized in RDF[2] files. For exam-

[1] OWL reference: http://www.w3.org/TR/owl-ref
[2] RDF reference: http://www.w3.org/TR/rdf-ref

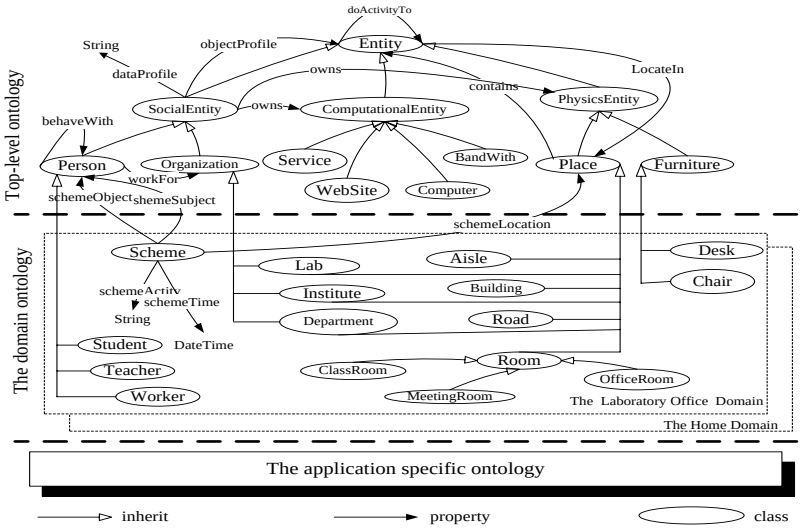

Fig. 1: Part of Our Ontology

ple, persistent context triple (Tom, type, Teacher) is a piece of RDF file like Fig. 2. Dynamic contexts are messages containing both RDF messages and other descriptive information. Dynamic context octuple (Tom, giveLecture, Room305, 15s, 116943354000, 116943388123, 2, Updated) is implemented like Fig. 3. This means context "Tom gives a lecture in Room305" is updated for the 2nd time at the UNIX time 116943588123, and if it doesn't be updated in the next 15 seconds, its state will become "Inert".

```
<rdf:resource rdf:ID= "Tom">
        < rdf:type  rdf:about="Teacher"/>
</rdf:resource>
```

Fig. 2: The Serialized Format of A Persistent Context

```
Ttl=15s                    Updatestime = 1116943388123
<person rdf:resource= "Tom">
        <giveLecture rdf:resource="Room305"/>
</person>
Frequency = 2              State = "Updated"
Starttime = 1116943354000
```

Fig. 3: The Serialized Format of A Dynamic Context

4 Context Management Mechanism

4.1 Context Reasoning

For reasoning high-level semantic contexts, we apply rule based reasoning and ontology based reasoning orderly on low-level contexts. The rules for reasoning are just horn

clauses for the consideration of system performance. Whereas the process is similar to [3], some significant improvements are brought in.

Firstly, time information is added to high-level contexts during inferences because this information is obviously important toward timely and accurate contexts. But how do we know exactly the starttime, updatetime and ttl value of each high-level inferred context? There are only intersection operations among a horn clause formed rule's antecedents without negation and union operations so that if a premise context become demoded, the inferred context should also correspondingly disappear from the system. Therefore, an approximate approach is implemented. When a high-level context $context_i$ is inferred by raw contexts and a rule, we select the earliest dying one which has the smallest value of ttl plus updatetime from $context_i$'s premises corresponding with the rule, and finally set the $context_i$'s ttl and updatetime the same as the selected one. The default starttime, state and frequency of $context_i$ are respectively set as its updatetime, "Beginning" and 1.

Secondly, the two reasoners are configured as traceable. Because derivation information is often needed for both judging which contexts will be discarded during inconsistency resolution and preventing future conflicts, the reasoning process is stored in the memory until a inconsistency resolution algorithm is performed.

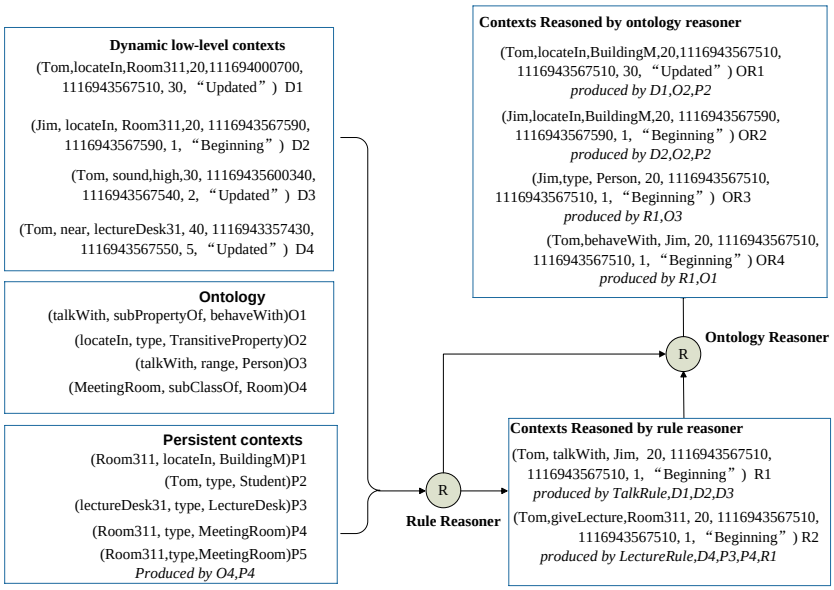

Fig. 4: Context Reasoning Example

An example of our modified reasoning flow is shown in Fig. 4. According to the first modification, inferred context R_1,R_2,OR_1, OR_3, OR_4's ttl and updatetime are set the same as D_1, the similar processes are applied on high-level contexts OR_2.

4.2 Context Lifecycle Management

In our prototype, an enhanced context lifecycle management strategy is carried out for every dynamic context. As described in Section 3, dynamic contexts have 4 life states: "Beginning", "Updated", "Inert", and "Disappearing". "Beginning" denotes the context is newly generated and no other replicas already exist in the system. "Updated" means the context has been refreshed recently. The design consideration for state "Inert" is to have those contexts which are existing in the real world but delayed in computer systems accidently due to either weakening of sensors' physical signals or bottlenecks of software infrastructures live for a little while. "Disappearing" means the context disappears in computer's view and will be discarded after a short period. It is obvious that using lifecycle states can make computers' contexts more timely and accurate so as to largely approximate real world contexts. Another intention for employing context lifecycle is that applications needs contexts depending on not only their contents but also their life status, for example, an application may open slides at the beginning of a seminar(exploiting beginning contexts) while close the slide at the end of the seminar(requiring disappearing contexts).

The 9 context life state transitions in the system are shown in Fig. 5. Transition 0, 1, 3, 5 and 8 are invoked for the reason that there are new contexts generated, either low-level ones from sensors or high-level ones from reasoning. The pseudocode for those transitions is shown as follow.

---

a new context $context_{new}$ is generated.
if $\exists$ $context_e$ in memory, $context_e$ has the same S-P-O triple with $context_{new}$
  if $context_e.state ==$ "Disappearing"
    $context_{new}$ substitute $context_e$
    (Transition 8 is performed)
  else
    $context_e.state =$"Updated",
    $context_e.updatetime = context_{new}.updatetime$
    $context_e.frequency = context_e.frequecny + 1$
    discard $context_{new}$
    (Transition 1 or 3 or 5 is performed)
else
  add $context_{new}$ into memory,
  (Transition 0 is performed)

A background thread runs periodically to tick the life period for every live context. When a "Beginning" or "Updated" context $context_i$'s ttl is no more than zero, its state turns to "Inert" (transition 2 or 4). After a fixed time, if $context_i$ is still not refreshed, it will become "Disappearing" and removed to historical context storage (transition 6 and 7). We store demoded contexts in persistent storage rather than discard them because historical contexts may be useful for various applications.

In practice, we found that using this lifecycle management can greatly abridge the gap between computers' contexts and real world's . Besides, context exploitation will become easier and more unambiguous.

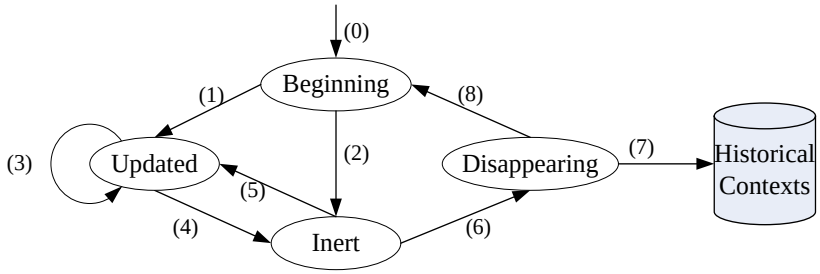

Fig. 5: State Transitions of Dynamic Contexts

4.3 Context Exploitation

Context Query. In our prototype system, we use RDQL[3] as context query language. But we extend RDQL for the particular features of contexts. Applications can query contexts by specifying a RDQL sentence with a state of contexts. For instance, we can use sentence "select ?x where (?x giveLecture Room311)(?x Type Teacher), Beginning" to search if there is a teacher who begins giving a lecture in Room311. Also, we can look up historical contexts conveniently by attaching time ranges to RDQL sentence.

Context Callback. Applications can exploit contexts not only by querying but also by registering callbacks. However, context callback mechanism should be much different from conventional event-callback mechanisms due to particularity of contexts. Contexts are varied with time and callbacks must exactly match to real world's requirements. For example, if a context-aware application's function is to open slides for lecturers automatically, with a badly designed callback mechanism, the application may open the slides more than once so that users are confused. Focusing on this, we invokes callbacks after every inferences and time tick, and use a replica pool to store consumed context for every applications respectively. When the callback is being invoked, the system check each replica pool and do not call those stored consumed contexts' callback function. Unless those consumed contexts have some changes, they will not be cleared out of every replica pool. This device embraces the particularity of contexts and leads to jarless applications in practice. The view of callback architecture is shown in Fig. 6.

5 Context Inconsistency Resolution

5.1 Conflict Detection

For inconsistency resolution, the first step is to detect conflicts. Ontology based context model can largely facilitate conflict detection. For example, if there are 2 dynamic contexts: d_1(Tom, giveLecture, Room311, 15s,1116943120489, 1116943567511, 10, Updated) and d_2(Tom, giveLecture, Aisle3, 25s, 1116943111897, 1116943567599, 1, Beginning), 2 persistent contexts: p_1(Room311, type, Room) and p_2(Aisle3, type, Aisle),

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

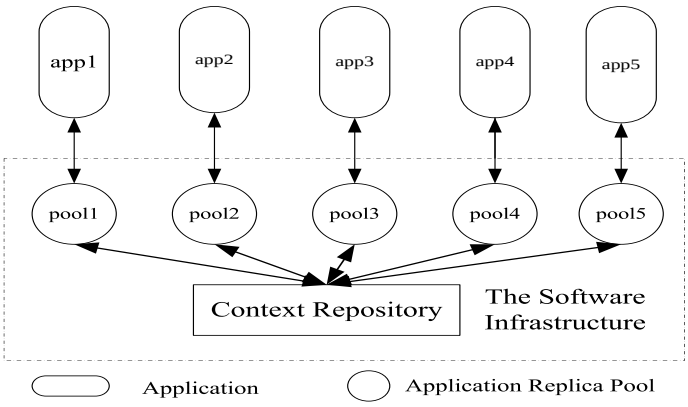

Fig. 6: The Callback Architecture

and 2 assertions in ontology: o_1(Room, disjointWith, Aisle) and o_2(giveLecture, type, FunctionalProperty), a conflict will be detected in ontology model because there is an instance of both Room and Aisle. However, d_1's derivations and d_2's usually implicitly conflict, therefore we need to find their derivations and resolve them completely in order to prevent future conflicts. Most semantic web APIs support conflict detection like that, and a validity report which indicate all first-hand conflicting pairs such as (d_1, d_2) will be easily obtained.

5.2 Several Definitions

Conflict pair set. A set consisting of pairs such as ($context_a$,$context_b$) that $context_a$ conflicts with $context_b$ is a conflict pair set.

Conflict set. Imaging a context set $ContextSet$, if its members are conflicting with each other, we call $ContextSet$ a Conflict set.

Derivation. If $context_a$ is a premise of high-level $context_b$, then we call $context_a$ is one of $context_b$'s Derivation. Furthermore, the relationship of Derivation is transitive and reflexive.

Derivation set. All of $context_c$'s Derivations compose a set called $context_c$'s Derivation Set.

Relative frequency$-rf$. A formula that calculates the rf value of a context $context_i$ is shown as follow.

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

5.3 CIR—Context Inconsistency Resolution Algorithm

The CIR(Context Inconsistency Resolution) algorithm is shown below.

---

1. Initialize
1). obtain a firsthand conflict pair set CFS from conflict detection results.
2). for every pair (a,b) $\in CFS$
    add both a and b into set $allContext$
3). for every $context_i \in allContext$
    a. Construct its derivation set $derivations_i$
    b. Construct $dynamicderivations_i$ which only contains dynamic contexts in $derivations_i$
2. Discard Contexts
while there are conflicts in $allContext$
  1). partition $allContext$ into several maximum conflict sets.
  2). for every conflict set $conflicts$
      select a context $context_{max}$ with largest rf.
      for every $context_i \in conflicts$ ($i \neq max$)
        for every $context_j \in dynamicderivation_i$
          if $\exists k, i \neq k$, $context_j \in dynamicderivation_k$
            reserve $context_j$
          else
            discard $context_j$
        delete $dynamicderivation_i$

Our design principle is that more frequent dynamic contexts are prior. However, different sorts of contexts are hard to compare their frequencies. For example, voice contexts may be inherently varied more frequently than temperature contexts, but we can't say that voice contexts have more priorities. Due to this reason, we use the rf value to measure each context's relative frequency because ttl value may often imply the context is inherently frequent or infrequent. We believe that those contexts with larger rf value emerge more relatively frequently recently, therefore they are more possible to be correct contexts. Also, persistent contexts are ensured to be consistent when they are been deployed to the platform so that they are always reserved.

It is ensured that after the algorithm, there is no conflict existing in the context repository. The step of partitioning $allContext$ uses a greedy algorithm, in which we begin to search from a random element, and form a maximum conflict set circularly until the partition is completely formed. Although the worst case time complexity of CIR is polynomial with the number of total contexts, we found in experiments that it is still such an expensive task that we can only run it periodically.

5.4 Example

Fig. 7 shows an example of the inconsistency resolution algorithm. In the example, we have two conflict sets: conflict set A and conflict set B. We first resolve conflicts for

A, and then for B. Assume that in A, $context_1$, $context_2$, $context_3$ and $context_4$ are ordered by their rf value increasingly. After A is resolved by the algorithm, there are 3 contexts$-$$context_4$, $context_{D4}$, $context_{D6}$ left.

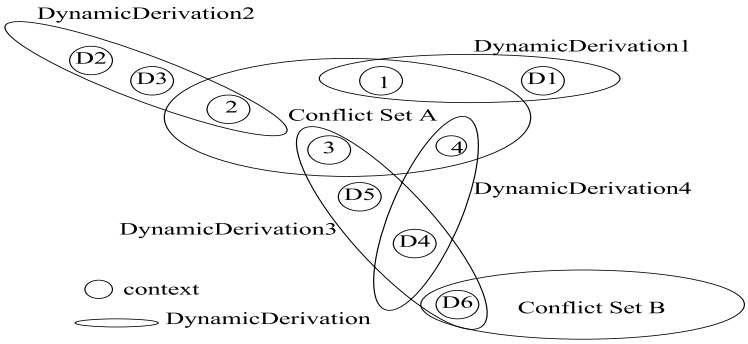

Fig. 7: An Example of the Inconsistency Resolution

6 Evaluations

During implementations, the semantic web API we choose is Jena2.2[4], the rules are in the form of Jena Generic Rules, and the ontology reasoner we used is entailed by OWL-Lite. We have modified Jena source code by adding time information to triples during reasoning, as described in section 4. The performance and effect of CIR algorithm are evaluated by 2 experiments.

First, we test the performance of CIR on a Linux Workstation with 4G RAM and 2 Xeon CPUs, and find that the efficiency is decreasing proportionally to the increasing of total contexts in memory. At the level of 1000$-$2000 contexts, the time used is 1.5 seconds$-$2.0 seconds, but at the level of 3000-4000, about 6 seconds are needed.

Second, for evaluating the effect of CIR, another experiment is designed. There are 3 computers involved, one Linux workstation with 4G RAM and 2 Xeon CPUs and two PC clients, connecting through LAN. The meeting room and aisle for the experiment are equipped with mica sensors[5] to detect noise and cricket sensors[6] to find persons' locations. One of the clients plays the role of raw context provider while the other acts as context consumer. In the experiment, a person adorning a cricket beacon stands in a meeting room to act as giving a lecture, and during this, he/she goes out to the aisle with immediately coming back to the meeting room at different frequencies which vary from 10s once to 40s once at the step of 5s (horizontal axis in Fig. 8), and maintains each frequency for 10 minutes. This activity can lead to many context conflicts among

[4] Jena2 Semantic Web Toolkit: http://www.hpl.hp.com/semweb/jena2.htm
[5] The Mica Sensor: http://www.xbow.com
[6] The Cricket indoor location system: http://cricket.csail.mit.edu/

high-level contexts in the system because two raw context triples: ($person_x$, locateIn, $MeetingRoom_x$) and ($person_x$, locateIn, $Aisle_x$) are obtained. Meanwhile, the context consumer client continues querying contexts 10 times a minute to see the probability of context correctness(vertical axis in Fig. 8). In this way, for every going out frequency, 100 samples about context quality can be gained. It is apparent that with the decreasing of the person's going out frequency, the incorrectness and inconsistency of contexts will decline. For comparing the effect with other solutions, 3 configurations of the context fusion infrastructure are carried out respectively: without any inconsistency resolution(without IR), with a simplistic resolution strategy that later updated and persistent contexts are prior(with SIR), with our proposed algorithm CIR(with CIR). The results are shown in Fig. 8.

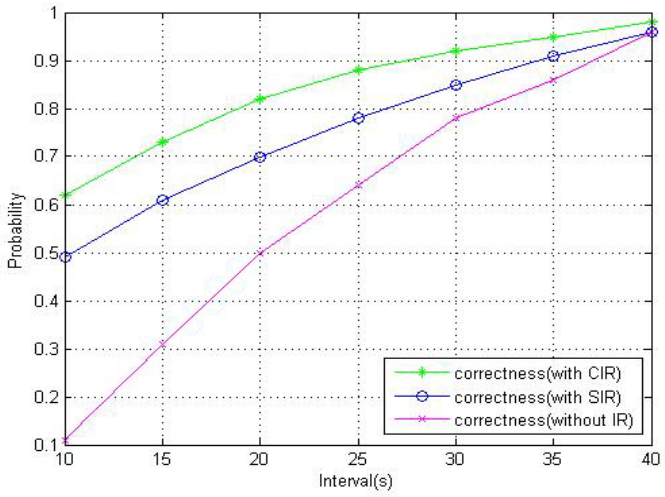

Fig. 8: Effect Analysis of the Inconsistency Resolution

Hence, although CIR is a computational intensive task, it is still necessary to run periodically. Although there are only 2 types of sensors used in the experiments, our architecture and algorithm can suit to more sensor types without modifications because they are designed for semantic contexts rather than physical sensors. And the only thing we need to do is to add specific raw context providers when new sensor types are brought in.

7 Application Case Study

7.1 Scenario

In research groups, seminars are often held. When someone gives a lecture, he/she should copy the slides to his/her flash disk, carry it to the meeting room, copy the

slides to the computer in the meeting room, and then open them. The work is dull and trivial, and many of people's attentions are consumed. In our context-aware computing environment, the lecturer needs to do nothing other than edit his/her lecture notes. When he/she enters the meeting room, and stands near the lectern, his/her slides will be opened automatically. During the seminar, if some strangers come in, a warning balloon will pop up on the screen. At the end of the seminar, the slides will be closed automatically.

7.2 Implementation

We implement two versions of the scenario, one of which is based on our context management mechanism(with CIR), the other of which is based on an earlier version which employs a simplistic inconsistency resolution strategy that later updated and persistent contexts are prior(with SIR).

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 it serves will give a lecture in the next few days, it will upload the slides he has edited recently, 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 detecting if strangers come in. When the Meeting Assistant detects the context that the lecturer leaves the room, it will close the slides. In this application, we've used the in-door location sensor Cricket to detect a person's location in a room, and also the Mica sensor to detect the noise in a room. Fig. 9 shows the runtime action of Seminar Assistant when a stranger comes into the meeting room during a seminar(a warning balloon is popped up). Part of the context reasoning process for this example is already shown in Fig. 4.

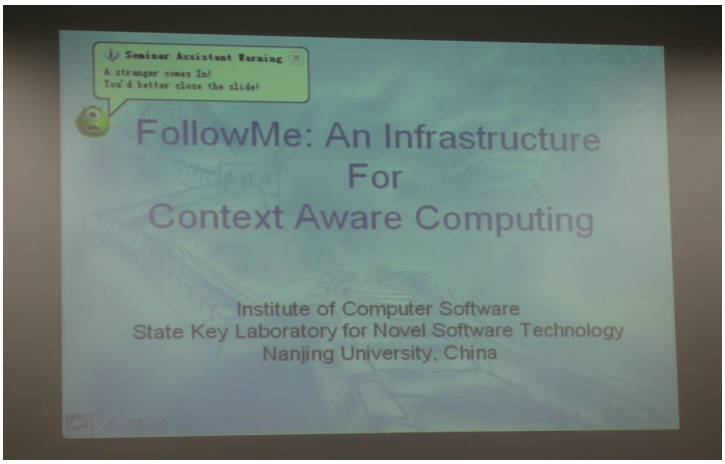

Fig. 9: The Runtime Effect of Seminar Assistant

7.3 Application Error Rate Comparison

We compare the two versions of “Seminar Assistant” by investigating their average error rates. Since both of the applications are very small, they are debugged exhaustively before our error rate comparison so that most errors occurring in the comparison should attribute to context mismatching. For the comparison, we run the two applications respectively for 20 days, use them according to the scenario for 400 times(20 times each day), and record the error rates of 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. 10 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 CIR) has largely improved context-aware applications’ robustness since over 50 percent incongruous behaviors are reduced(from 33 errors of 400 to 16 errors of 400).

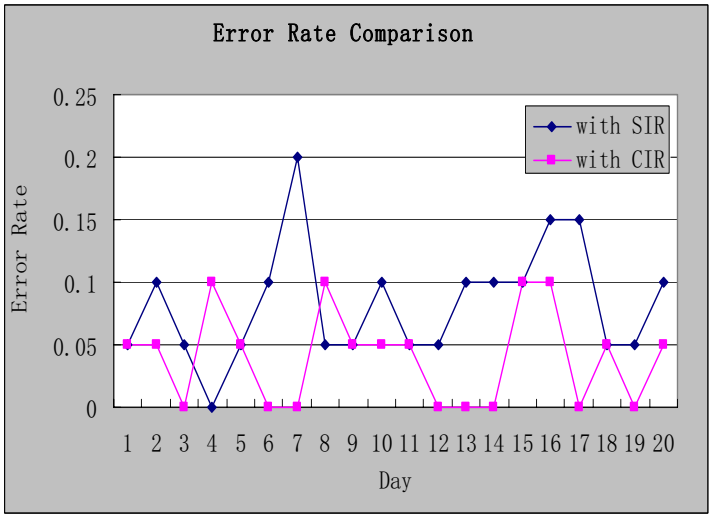

Fig. 10: Error Rate Comparison of The Two Versions

8 Conclusions and Future Work

With experiences of developing context-aware applications, we find that the inconsistency of contexts is a serious problem which can threaten the prevalence of context-aware computing. Aiming at this problem, we propose an extended ontology based context model called pvCM, establish a context management mechanism and design an inconsistency resolution algorithm. Through the evaluations and case study, the necessity and feasibility of our design principles are verified. The work of this paper

is part of our ongoing research project−FollowMe [6] which is designed towards a workflow-driven, service-oriented, pluggable and programmable software infrastructure for context-awareness.

In the near future, we plan to explore novel approaches to improve runtime performances of context reasoning, using technologies such as distributed context fusion and so on. Also, we are working towards a better inconsistency resolution approach in which context conflicts are resolved during the reasoning process, with sophisticated reasoning technologies.

References

1. Joelle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of ACM*, 48(3):49–53, 2005.
2. Harry Chen, Timothy W. Finin, Anupam Joshi, Lalana Kagal, Filip Perich, and Dipanjan Chakraborty. Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 8(6):69–79, 2004.
3. Xiaohang Wang, Jin Song Dong, ChungYau Chin, SankaRavipriya Hettiarachchi, and Daqing Zhang. Semantic space: An infrastructure for smart spaces. *IEEE Pervasive Computing*, 3:32–39, July-September 2004.
4. Tao Gu, H. K. Pung, and Da Qing Zhang. Towards an osgi-based infrastructure for context-aware applications in smart homes. *IEEE Pervasive Computing*, December 2004.
5. Yingyi Bu, Jun Li, Shaxun Chen, Xianping Tao, and Jian 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.
6. Jun Li, Yingyi Bu, Shaxun Chen, Xianping Tao, and Jian 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.
7. Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.
8. Daniel Salber, Anind K. Dey, and Gregory 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.
9. Jason I. Hong and J.A. Landa. An infrastructure approach to context-aware computing. *Human-Computer Interaction (HCI) Journal*, 16, 2001.
10. Guanlin Chen. Solar: Building a context fusion network for pervasive computing. *Ph.D. Thesis. Dartmouth College*, August 2004.
11. Harry Chen, Timothy W. Finin, and Anupam Joshi. Semantic web in the context broker architecture. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), 14-17 March 2004, Orlando, FL, USA*, pages 277–286, 2004.
12. Tim Kindberg and John J. Barton. A web-based nomadic computing system. *Computer Networks*, 35(4):443–456, 2001.
13. Gregory Biegel and Vinny 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.

14. Fahy P. and Clarke S. Cass: Middleware for mobile context-aware applications. In *ACM MobiSys Workshop on Context Awareness, Boston, USA*, 2004.
15. Lonnie D. Harvel, Ling Liu, Gregory D. Abowd, Yu-Xi Lim, Chris Scheibe, and Chris 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.
16. Xiaodong Jiang, Nicholas Y. Chen, Jason I. Hong, Kevin Wang, Leila Takayama, and James 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.
17. Pauline P. L. Siu, Nalini Moti Belaramani, Cho-Li Wang, and Francis C. M. Lau. Context-aware state management for ubiquitous applications. In *Proceedings of International Conference on Embedded and Ubiquitous Computing*, pages 776–785, 2004.
18. T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, September 2004.
19. Anupama Kalyan, Srividya Gopalan, and V. Sridhar. Hybrid context model based on multilevel situation theory and ontology for contact centers. In *Proceedings of the 3rd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops), 8-12 March 2005, Kauai Island, HI, USA*, pages 3–7. IEEE Computer Society, 2005.
20. Maria Strimpakou, Ioanna Roussaki, Carsten Pils, Michael Angermann, Patrick Robertson, and Miltiades E. Anagnostou. Context modelling and management in ambient-aware pervasive environments. In *Proceedings of First International Workshop on Location- and Context-Awareness (LoCA), Oberpfaffenhofen, Germany, May 12-13, 2005*, volume 3479 of *LNCS*, pages 2–15. Springer, 2005.
21. Anind K. Dey and Jennifer Mankoff. Designing mediation for context-aware applications. *ACM Transactions on Computer-Human Interaction(TOCHI)*, 12(1):53–80, 2005.
22. Chang 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.
23. Chang Xu, Shing-Chi Cheung, and W. K. Chan. Incremental consistency checking for pervasive context. In *Proceedings of the 28th International Conference on Software Engineering(ICSE 2006), Shanghai, China, May 20-28, 2006*, pages 292–301, 2006.
24. Jussi Myllymaki and Stefan 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.
25. Gadia and Sunil S. Nair. Temporal databases: A prelude to parametric data. In *Temporal Databases*, pages 28–66. 1993.