An Enhanced Ontology Based Context Model and Fusion Mechanism*

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

State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing City, P.R.China, 210093
{byy, lijun, csx, txp, lj}@ics.nju.edu.cn

Abstract. With diverse sensors, context-aware applications which aim at decreasing people's attentions to computational devices are becoming more and more popular. But it is inadequate just with sensors because what applications really need is high-level context knowledge rather than low-level raw sensor data. So a software layer which delivers high quality contexts to applications with an easy application programming model is needed. In this paper, we establish a formal context model using semantic web languages and design a context fusion mechanism which not only generates high-level contexts by reasoning, but also brings in context lifecycle management and periodically time based conflict resolution to improve the quality of contexts. Using the context fusion mechanism, a programmable and service oriented middleware is built upon OSGi framework to support context-aware applications. Also, we propose an application programming model using RDQL as context query language and demonstrate an application called Seminar Assistant. According to the experiment results, we believe our prototype system is useful for non-real-time applications in various domains.

1 Introduction

Compared with desktop computing, the most remarkable advantage of ubiquitous computing is context-awareness. Context refers to any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) [1]. Computational entities in ubiquitous computing environment need to be context-aware so that they can follow changing situations and provide personalized services according to different situations. In this way, people's attentions to diverse computational devices can be largely decreased so that computers may "weave themselves into the fabric of everyday life until they are indistinguishable from it" [2].

To achieve context-awareness, applications should obtain semantic contexts meeting their needs. However, what sensors provide are just low-level physical data which have no semantic meanings so that applications should convert those sensor data to high-level context knowledge themselves. We think the task of fusing contexts should shift to a shared infrastructure which decouples applications with sensors and context fusion mechanism, provides high quality contexts, and fosters ease of programming.

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

For solving these problems, we establish an enhanced ontology based context model using semantic web languages and design a context fusion infrastructure not only generating high-level contexts formally but also making contexts timely, accurate and consistent. With dynamic deploying of domain contexts, the infrastructure can be easily applied to different domains. Also, we provide an easy application programming model and develop a small context-aware application−Seminar Assistant. To evaluate the feasibility of our infrastructure, we give a performance study.

The rest of this paper is organized as follows. Section 2 discusses on related work. In Section 3 we present our context model. The context fusion mechanism is presented in Section 4. The infrastructure is introduced in Section 5. The performance study is shown in Section 6. Section 7 presents our application programming model and an example. Finally, we conclude in Section 8.

2 Related work

In previous work, researchers have proposed many context models such as key-value, XML, object, UML-ER, Ontology and so on [3], and fused contexts formally or informally.

Among Systems which use informal fusion mechanism, the typical one Context-Toolkit [1] established an object-oriented framework and a number of reusable components; Cooltown [4] proposed a web-based context-aware system; Context Fabric [5] defined a Context Specification Language and a set of core services. But all of them lack a general context representation and fuse different contexts in different ways. Although ubi-UCAM [6] uses a formal unified 5W1H (Who, What, Where, When, Why and How) object-based context model, it fuses contexts informally. Because of informal context fusion, those frameworks and infrastructures are difficult to reuse.

Formal context fusion mechanism can generate high-level contexts for different applications in the same way so that the common module for fusing contexts can be extracted as a shared, highly reusable infrastructure. Karen [7] modeled contexts using both ER and UML models. Anand [8] represented contexts in Gaia system as first-order predicates written in DAML+OIL[1]. CoBrA [9] established an ontology based context model and an agent-based system for smart room environment. SOCAM [10] proposed an OWL[2] ontology based context model addressing context sharing, reasoning and knowledge reusing, and built a service oriented middleware infrastructure for applications in a smart home. However, none of them considered the lifecycle management of contexts or introduced their conflict resolution principles. And, none of the infrastructures can be easily customized for different domains. Also, they didn't give us a clear and easy application programming model.

3 Context Model

A good context model can lead to a well designed context fusion mechanism. The ideal context model which can make contexts easily shared by different applications should

[1] DAML+OIL reference: http://www.w3.org/TR/daml+oil-reference
[2] OWL reference: http://www.w3.org/TR/owl-ref

also enable context fusion both to make high-level contexts more timely, exact, complete, conflict-free and to evolve easily.

To achieve this ideal context model, our context model consists of 2 parts: ontology and its instances. The ontology which is a set of shared vocabularies of concepts and the interrelationships among these concepts enables context sharing, logic reasoning and knowledge reusing [10]. Instances of ontology include both persistent contexts and dynamic contexts. Persistent contexts usually have a long life period and they would be combined with dynamic contexts during reasoning process. Triples described as (subject, predicate, object) are used to model persistent contexts. For example, the context "Tom is a teacher" is modeled as (Tom, type, teacher). Dynamic contexts with transient characteristics only have a short life in the system, such as "Tom in Room311". Quintuples (subject, predicate, object, ttl, timestamp) are used to model them. Ttl denotes the life period of a context while timestamp means the UNIX time when the context is generated or updated.

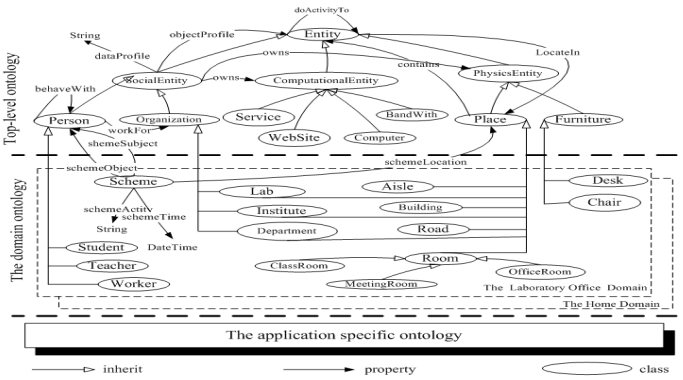

Fig. 1: Part of Our Ontology

In our implementation, the ontology is modeled by OWL-Lite. Expressive descriptions such as "TransitiveProperty", "SymmetricProperty", "Disjoint" and so on in OWL-Lite can largely decrease the number of user-defined rules. We construct our ontology in a hierarchical manner, fostering knowledge reuse. From top to down, there are top-level ontology, domain ontology and application specific ontology. Compared with CoBrA [9] which combines activities and entities as classes like "PeopleInMeeting" and SOCAM [10] which describes most activities as classes like "Talk", our ontology uses properties to represent activities for ease of describing contexts uniformly in triples or quintuples. Fig. 1 shows part of our ontology which is specialized for the laboratory office domain. Persistent contexts are serialized as RDF[3] files while dynamic contexts are implemented as RDF messages attached with ttl and timestamp.

[3] RDF reference: http://www.w3.org/TR/rdf-ref

4 Context Fusion Mechanism

Upon the context model, our context fusion mechanism is logic inference based. We apply rule based reasoning and ontology based reasoning orderly on raw contexts to generate high-level contexts, using the Jena API[4]. The user-defined rules are in the form of Jena generic rules without negation and "or" operation. For separating context generation and applications, we use deductive rules rather than reactive rules. Fostering convenient lifecycle management and conflict resolution, we modify the Jena source code and add time information to contexts during reasoning.

4.1 Adding Time Information During Reasoning

Time information is added to dynamic high-level contexts during reasoning. In detail, when a high-level context $context_{conclude}$ is generated, we have its premise context set $Facts$: $\{context_1, context_2, \ldots, context_n\}$, which has a subset $DynamicFacts$ consisting of dynamic contexts: $\{dyncontext_1, dyncontext_2, \ldots, dyncontext_k\}$. First, we select $context_{min}$ which has the minimum value of ttl plus timestamp in DynamicFacts. Then we set the ttl and timestamp of $context_{conclude}$ the same as $context_{min}$'s. The princeple here is to make the ttl and timestamp of every high-level context the same as the earliest dying one in its premise set. The reason is that there are only "and" operations to connect premises in Jena generic rules and a high-level context will become demoded when whichever of its premises is demoded. Because the operation of adding time restriction is embedded in every inference operation, there is little additional time used.

4.2 Reasoning High-level Context

When we infer high-level contexts, rule reasoner and ontology reasoner are used orderly. The two reasoners are configured as traceable in order to facilitate conflict resolution, though much more memory is used.

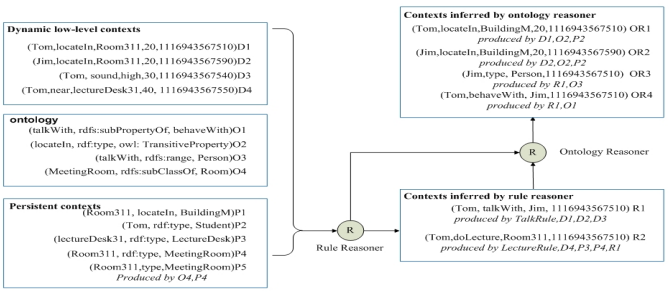

Fig. 2: An Example of Context Reasoning

[4] Jena2 Semantic Web Toolkit: http://www.hpl.hp.com/semweb/jena2.htm

Here we take an example to explain the reasoning process. As Fig. 2 shows, in the example, we have 4 dynamic low-level contexts, 3 (D_1, D_2, D_4) of them coming from Cricket[5] related context provider while another (D_3) coming from Mica sensor[6] related context provider. Also, 4 assertions (O_1, O_2, O_3, O_4) in ontology and 4 persistent contexts (P_1, P_2, P_3, P_4) have been deployed to the platform. But persistent context P_5 is deduced from O_4 and P_4 when (P_1, P_2, P_3, P_4) are being deployed to the platform. The two user-defined rules used in this reasoning instance are shown as follows:

$\mathbf{TalkRule}: \quad (?x \quad locateIn \quad ?room), (?y \quad locateIn \quad ?room), (?room \quad rdf: type \quad Room), (?x \quad sound \quad high) -> (?x \quad talkWith \quad ?y)$
$\mathbf{LectureRule}: \quad (?x \quad locateIn \quad ?room), (?room \quad rdf: type \quad MeetingRoom), (?x \quad near \quad ?lectern), (?lectern \quad rdf: type \quad Lectern), (?x \quad talkWith \quad ?y) -> (?x \quad doLecture \quad ?room)$

We can see that after reasoning, high-level contexts $\{R_1, R_2, OR_1, OR_2, OR_3, OR_4\}$ are generated.

4.3 Context Lifecycle Management

With time information for contexts, a lifecycle management is brought in for dynamic contexts. Fig. 3 shows context lifecycle.

A background thread runs periodically to tick the life period for every live context. When a context's ttl is no more than zero, it is demoded and removed to historical context storage. We store demoded contexts in persistent storage rather than discard them because historical contexts may be useful for various applications. When a new generated context has the same (subject, predicate, object) as an existing one, we update the older one by making the newer displace it. Applications can register callbacks not only by specifying interested contexts but also by pointing out which transition (generated, updated or demoded) they want.

Through lifecycle management, demoded contexts have been removed away so that context conflicts are fewer than normal. The contexts for applications can be more accurate and timely.

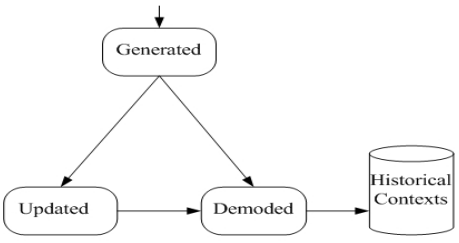

Fig. 3: Lifecycle of Dynamic Contexts

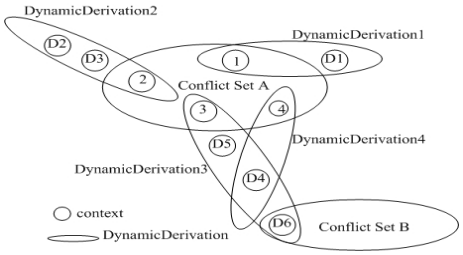

Fig. 4: An Example of Conflict Resolution Algorithm

[5] The Cricket indoor location system: http://cricket.csail.mit.edu/
[6] The Mica Sensor: http://www.xbow.com

4.4 Conflict Resolution

Inconsistent contexts are often emerging in systems due to faults of either physical sensors or software. Using ontology can facilitate conflict detection. For example, if our context repository has 2 dynamic contexts: (Tom, locateIn, Room311, 15, 1116943567511) and (Tom, locateIn, Aisle3, 15, 1116943567599), 2 persistent contexts: (Room311, rdf:type, Room) and (Aisle3, rdf:type, Aisle), and 2 assertions in ontology: (Room, owl:disjointWith, Aisle) and (locateIn, rdf:type, owl:FunctionalProperty), a conflict will be detected in ontology model because there is an instance of both Room and Aisle. But how to resolve this conflict and make contexts consistent? We've designed an algorithm to resolve context conflicts, which is based on the time factors of contexts. Our design principle is that the later contexts and persistent contexts have more priorities.

From the Jena API, a validity report can be obtained, which indicates every first-hand conflicting pairs. Then we can easily construct several conflict sets whichever consists of contexts conflicting with each other, from the validity report. For every context $conflictcontext_i$ in every conflict set, we trace its derivation and make a set called $DynamicDerivation_i$ which contains dynamic contexts in $conflictcontext_i$'s derivation set and $conflictcontext_i$ itself. For the example in Fig. 2, R_2(Tom, doLecture, Room311, 1116943567510) is a high-level context and its $DynamicDerivation$ is $\{D_1, D_2, D_3, D_4, R_1, R_2\}$. Then we resolve conflicts for every conflict set orderly.

Now we explain in detail how to resolve conflict for one conflict set called $Conflicts$. We compare each context in the $Conflicts$. If two dynamic contexts conflict, we discard the earlier one's DynamicDerivation. But during discarding a DynamicDerivation, contexts which exist in other existing DynamicDerivations are reserved. As an exception, if there is any persistent context in $Conflicts$, all other contexts' DynamicDerivation should be discarded. It is ensured that there is no conflict among persistent contexts because when they are deployed to our infrastructure, we check the consistency and reject conflicting ones.

Fig. 4 shows an example of the conflict 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 timestamp increasingly. After A is resolved by the algorithm, there are 3 contexts$-$$context_4$, $context_{D4}$, $context_{D6}$ left.

It is obvious that through the conflict resolution, there is no conflict left in context set although some right contexts may be discarded. Nevertheless, contexts are different from general knowledge in the way that correct contexts are generated frequently while wrong contexts emerge by accident. Therefore, correct contexts will appear soon even if they are wrongly deleted while the probability of wrong contexts' repetitious appearances is very low. However, the conflict resolution algorithm is a computational intensive task so that we make it run periodically.

5 The Middleware Infrastructure

Based on this context fusion mechanism, we've build a centralized OSGi[7] based context service as an infrastructure to accept raw contexts from context providers, produce high-

[7] OSGi, Open Service Gateway Initiative: http://www.osgi.org

level contexts, perform context lifecycle management, resolve context conflicts, and deliver contexts to all applications. In the system, besides the context service, there are following roles: sensors which gather data from the physical world, such as location, temperature, pressure, picture, noise and so on; context providers which interact with sensors, and transform sensor data into raw semantic contexts; and applications (context consumers) which consume contexts by either subscribing or querying, and adapt to context changes.

In our smart environment, sensors, context providers, context consumers and the central context service are physical distributed. Computers connect through a local network. To ease the communication among different nodes, we use Java RMI technology. One feature of the infrastructure is dynamic deploying of ontology, persistent contexts and rules without restarting the service. With the help of domain independence of the context service, application developers of our infrastructure can easily customize the context service to their specific domains by deploying the domain related ontology, persistent contexts and rules. In our prototype, we customize the infrastructure to support the laboratory office domain.

6 Evaluations of Context Service

For evaluating the context service, we've tested it in following aspects:

Performance of conflict resolution. The experiment has been conducted on a set of Linux workstations with different hardware configurations (512MB-RAM/P4-1.4 GHz, 1GB-RAM/P4-2.8 GHz, and 4G-RAM/2-XEON-CPU). The ontology reasoner we have used is entailed by OWL-Lite, and the rule reasoner applies a rule set consists of 8 forward-chaining rules. Fig. 5 shows the results. The performance of our conflict resolution algorithm is acceptable when a high-powered workstation is used.

Effect of conflict resolution. We use a simulating context provider in a client node which repeats sending two conflicting raw contexts orderly to the context service. In this way, many inconsistent contexts will occur in the system so that the consistency and accuracy of contexts can be tested. It is obvious that the interval between the two contrary contexts can largely influence the results so that we make the interval change from 10s to 40s at the step of 5s and maintain each interval for 10 minutes to carry out the test. An application in another client node queries contexts 10 times in every minute to see the probability of getting conflicting contexts and getting correct contexts for each interval. The central server is located at a Linux workstation which has 4G RAM and 2 Xeon CPUs. The experiment results are shown in Fig. 6 ("CR" means conflict resolution). We can see that with the time based conflict resolution, the quality of contexts is much better.

Performance of the infrastructure. The configuration of the experiment includes one server and two clients. The central server is also the Linux workstation with 4G RAM and 2 Xeon CPUs. A simulating context provider is running at one client node and sends raw contexts to the central server at different frequencies while an application is running at another client node and queries contexts from the central server twice a minute. Tests are carried out respectively with the number of total live contexts at 3 levels: $3000-4000$(suited for a smart home), $5000-6000$(suited for a small office) and

8000−9000(suited for a middle office). We record the latency of this application for each frequency within every total context range. From the results shown in Fig. 7, we can see that after a curve point in each line, the latency of acquiring contexts increases in an exponential speed.

From the results of our experiments, we can get several useful guides for design context fusion mechanism:

Firstly, our time based conflict resolution algorithm is an expensive task so that it is impossible to run it every time when high-level contexts are generated. But it is necessary to run the algorithm periodically to decrease the probability of conflicts and faults.

Secondly, this logic inference based context fusion mechanism isn't suitable for real-time applications. But it's still valuable for non-time-critical applications in domains such as office, home, hotel, school and so on.

Thirdly, the centralized design with a bottleneck existing isn't suitable for large smart space so that it is necessary to develop distributed context fusion mechanisms.

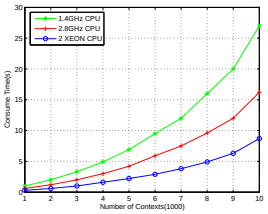

Fig. 5: Performance of Conflict Resolution

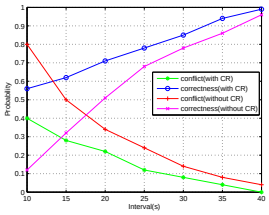

Fig. 6: Probability of Conflicts and Correctness

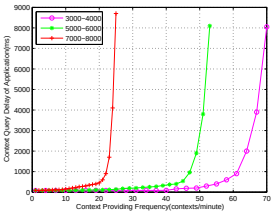

Fig. 7: Performance of The Infrastructure

7 Application Programming Model

7.1 How to Get Needed Context?

For querying contexts, we provide a RDQL[8] based context query mechanism. RDQL is a query language which can select matched RDF triples from a triple set. Applications based on our context service infrastructure can query contexts by specifying a RDQL query sentence.

Because the syntax of RDQL is complex, we use a simple example to illustrate how it works rather than introduce all of it. For example, there are 4 contexts in our context repository: A(Tom, type, teacher), B(Tom, doLecture, Room311,15, 1116943567511), C(Bob, locateIn, Room311, 15, 1116943567580) and D(Bob, type, Student). If we use a RDQL query sentence like that: "select ?x where (?x type Teacher),(?x locateIn

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

Room311)", we can get context A and B as the result. In this way, we find that RDQL can specify very complex contexts flexibly so that it is very suitable to be a context query language. Also, applications can get historical contexts by setting the time constrains in queries.

Based on this query mechanism, a callback interface is easily designed. Applications can register their interested contexts with wanted transitions (Generated, Updated or Demoded) to the context service at central server. If registered contexts occur and their transitions are suited, the applications will be notified.

7.2 Case Study

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.

Implementation. We implement the scenario based on the context service. 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. 8 shows a piece of the source code.

Seminar.java

```
public class Seminar implements ContextConsumer {
   ……
   public void callBackOnContexts(Iterator contexts) {
          Context context = (Context)contexts.next(); //get context
     String lecturer = context.getSubject().getName();
          openLectureNote(lecturer); //Open the Slides
   }
   public void register()
   {
     String query = "select ?x where (?x doLecture "+
              peoperties.getProperty("This-Room")+")";
          ContextFilter
            filter=contextFactory.createQLContextFilter(query,GENERATED)
            ;
   contextService.registerContextConsumer(this,filter);
    //register interested context to the ContextService
      }
…… }
```

Fig. 8: A Piece of The Code of Seminar Assistant

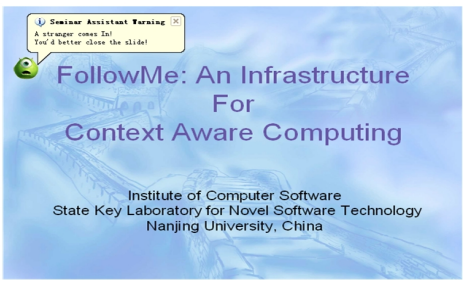

Fig. 9: The Runtime Effect of Seminar Assistant

Runtime Effect. Fig. 9 shows the runtime action of Seminar Assistant when a stranger comes into the meeting room during a seminar. Part of the context reasoning process for this example is already shown in Fig. 2.

8 Conclusion & Future work

Our study in this paper shows that our context fusion mechanism is feasible and necessary for providing context-aware applications with high quality contexts. We've implemented a context fusion service as an infrastructure to support context-aware applications, decoupling applications with sensors and context fusion. A case study shows that it's easy and rapid to develop applications based on our platform. The work of this paper is part of our ongoing research project−FollowMe which aims at a workflow-driven, service-oriented, pluggable and programmable context-aware infrastructure. Our next step is to explore novel approaches to both improve the quality of contexts and reduce the time cost. Also, we are working towards a distributed context processing mechanism to make context-aware systems more efficient and robust.

References

1. A. K. Dey, D. Salber, G. D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Anchor article of a special issue on Context-Aware Computing, Human-Computer Interaction (HCI) Journal, 16(2-4), 2001, 97-166.
2. Weiser. M. Computer for the 21st century. Scientific American, 265(3), 1991, 94-104.
3. T. Strang, C. Linnhoff-Popien. A Context Modeling Survey. Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, September 2004.
4. T. Kindberg, J. Barton. A Web-based Nomadic Computing System. Journal of Computer Networks. 35(4). 2001: 443-456.
5. J. Hong and J. Landay. An infrastructure approach to context-aware computing. Human Computer Interaction (HCI) Journal, 16(2-4), 2001: 287-303.
6. S. Jang, W. Woo. Ubi-UCAM: A Unified Context-Aware Application Model. In Proceeding of Modeling and Using Context − the 4th International and Interdisciplinary Conference. Springer. Stanford, CA, USA. June 2003. pp.178-189.
7. Karen Henricksen, et al. Modeling Context Information in Pervasive Computing Systems. First International Conference on Pervasive Computing. Springer. August 2002. Zrich, Switzerland. pp.167-180.
8. Anand Ranganathan, et al. A Middleware for Context-Aware Agents in Ubiquitous Computing Environmentse. In Proceeding of ACM/IFIP/USENIX International Middleware Conference. Springer. June 2003. Rio de Janeiro, Brazil. pp.143-161.
9. H. Chen, T. W. Finin, A. Joshi, L. Kagal, F. Perich, D. Chakraborty. Intelligent Agents Meet the Semantic Web in Smart Spaces. IEEE Internet Computing, (November 2004):69-79.
10. T. Gu, H. K. Pung, D. Q. Zhang. Towards an OSGi-Based Infrastructure for Context-Aware Applications in Smart Homes, IEEE Pervasive Computing, 3(4) 2004, 66-74.