Devising a Context Selection-Based Reasoning Engine for Context-Aware Ubiquitous Computing Middleware

Donghai Guan, Weiwei Yuan, Seong Jin Cho, Andrey Gavrilov, Young-Koo Lee, and Sungyoung Lee[*]

Department of Computer Engineering, Kyung Hee University, Korea
{donghai,weiwei,babebear,avg,sylee}@oslab.khu.ac.kr, yklee@khu.ac.kr

Abstract. We propose a novel reasoning engine for context-aware ubiquitous computing middleware in this paper. Our reasoning engine supports both rule-based reasoning and machine learning reasoning. Our main contribution is to utilize feature selection method to filter the low-level contexts which are not useful for certain special high-level context reasoning. As a result, rules and learning models in the reasoning engine's knowledge base are refined since useless context have been filtered. The merits of our proposed reasoning engine are described in details in this paper.

1 Introduction

By gathering context data and adapting systems behaviors accordingly, context-aware systems offer entirely new opportunities for application developers and end users. Especially when combined with mobile devices, these systems are of high value and are able to increase usability tremendously.

In order to adapt applications to different situations and be more receptive to users' needs, a number of works have been done in trying to make applications context aware in ubiquitous computing environments [1][2][3][4][5]. The design of these applications needs to take account of heterogeneous devices, mobile users and rapidly changing contexts. Hence, ubiquitous computing environments must provide middleware support for context-awareness.

The role of middleware is to ease the task of designing, programming and managing distributed applications by providing a simple, consistent and integrated distributed programming environment; such middleware-based approach is quite appealing in context-aware ubiquitous computing [2].

In last several years, we have developed our own context-aware ubiquitous computing middleware (CAMUS) [6]. In the process of analyzing existing middleware and devising our own middleware, we have identified a set of necessary functional elements that a context-aware system needs to support essential context aware mechanisms. These functional elements are shown in Table 1.

[*] Corresponding author.

Table 1. Necessary functional elements in context-aware middleware

Context Sensing	Obtain the context data from diverse context sources
Context Modeling	Foundation for expressive context representation and high-level context interpretation
Context Repository	Provide a persistent storage for distributed context
Context Reasoning/Inferring	Interpret low-level information and derive additional, high-level context
Context Discovery/Delivery	Searching appropriate context aggregators and delivering them to the applications.

Instead of describing all the elements in Table 1, in this paper, we focus on context reasoning. We argue that context reasoning plays an important role in a context-aware ubiquitous system. To truly understand the importance of context reasoning, first of all, we need to know what is context.

Context is defined as any information that can be used to characterize the situation of an entity. And entity could be the person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [7]. Context can be divided into low-level context and high-level context. In general, low-level context is simple and can be directly got from sensors or other sources. While, high-level context is abstract and need to be inferred from a piece of low-level context. This can be as simple as taking in a name and returning the corresponding email address. It could be more complex and take in the number of people in a room, the relative gaze directions, the audio level, and the time of day, and return whether or not a meeting was occurring.

Most applications show more interest in high-level context. For example, many smart spaces [8][9][10] would like to know the information about user's activity, which is a typical high-level context. Some adaptations will be made based on activity.

After understanding what is high-level context and why it is so important. Now we would like to present the reason why we should do the high-level context reasoning task in the middleware. Why not in application? There are two main reasons. One reason is that in ubiquitous environment, many applications are installed in mobile devices (mobile phone, PDA). Due to the constrained nature of small mobile computers in terms of processing power, memory and persistent storage, it is very difficult for them to do the reasoning task. Context reusability is the other reason to put the reasoning engine in middleware, not applications themselves.

This paper is organized as follows: in section 2, by introducing the related works about the reasoning engines adopted by other ubiquitous middleware, we conclude that all of them utilize rule-based reasoning and/or machine learning methods. This drives us to propose multiple reasoning mechanisms in our middleware's reasoning engine. In section 3, we present the main characters of our reasoning engine: multiple reasoning mechanisms, pluggable reasoning modules and ontology reasoning module. Section 4 gives our proposed method on how to improve reasoning engine based on our existing one. We propose to use feature selection method to filter the irrelevant low-level context for a given high-level context. At last, we present conclusions and future works in section 5.

2 Related Work

In this section, we will cover context reasoning mechanisms adopted by some well-known ubiquitous computing middleware.

In Context Toolkit [1], Context Interpreters are responsible for interpreting or converting context from one form to another. They maintain no state but simply take context in and output new context information. While the Context Toolkit does provide a starting point for applications to make use of contextual information, it does not provide any generic mechanism for writing rules about contexts, inferring high-level contexts or organizing the wide range of possible contexts in a structured format.

In Gaia [11], Context Synthesizer takes charge of reasoning. Context Synthesizers are agents that provide high-level contexts based on simpler sensed contexts. A Context Synthesizer gets source contexts from various Context Providers, applies some sort of logic to them and generates a new type of context. Agents can reason about context using rules written in different types of logic like first order logic, temporal logic, description logic, higher order logic, fuzzy logic, etc. Instead of using rules written in some form of logic to reason about context, agents can also use various machine learning techniques to deal with context. Learning techniques that can be used include Bayesian learning, neural networks, reinforcement learning, etc.

In SOCAM [12], the Context Reasoning Engine reasons over the knowledge base. Multiple logic reasoners can be incorporated into the Context Reasoning Engine to support various kinds of reasoning tasks. Currently RDFS reasoner, OWL reasoner and a general rule-based reasoner are built in to SOCAM. Different inference rules can be specified and preloaded into various logic reasoners. The interpreters is implemented by using Jena2 [13], a semantic Web toolkit. The Context Interpreter also acts as a context provider as it provides high-level contexts by interpreting low-level contexts. It consists of a context reasoner and a context KB.

In CASS [14], deriving of high-level context is also based on an inference engine and a knowledge base. The knowledge base contains rules queried by the inference engine to find goals using the so-called forward chaining techniques. As these rules are stored in a database separated from the interpreter neither recompiling nor restarting of components is necessary when rules change. In Cobra [15], the reasoning engine is responsible for reasoning with ontology knowledge and contextual knowledge in the knowledge base. In addition, the inference engine will apply learning algorithms and pattern recognition mechanisms to learn about high-level context. In ECA [16], the reasoning engine determines when the left-hand-side (LHS) of user specified rules are matched by the current set of facts stored in the knowledge base of the engine. As a result of this process, the engine initiates the actions specified in the right-hand-side (RHS) of rules.

By analyzing above context reasoning engines, we can see that although their detailed implementation parts are different, the main reasoning methods they applied are almost the same: rule-based reasoning and/or machine learning methods. Rule based reasoning is the most common form of knowledge processing nowadays. User predefined rules written in some form of logic are used to infer different contexts. However, most developers find that building the rules is the most difficult task in ubiquitous computing systems. Also rule-based reasoning is not flexible and can not adapt to changing circumstances. Making use of machine learning techniques to deduce the higher-level context enables us to get around this problem. In addition to multiple reasoning methods, knowledge base is also needed. The knowledge base is composed of two parts. One part is user-defined rules for rule-base reasoning. The other part is learning models. These learning models are inferred by utilizing inductive methods to analyze context history. In the next section, we will describe the detailed structure and important characters involved in our reasoning engine.

3 Our Existing Reasoning Engine

3.1 Overview

Our context reasoning engine includes one to many reasoners which handle the facts present in the repository as well as to produce composite contexts. The reasoners can provide the entailed knowledge not formally present in the repository using various kinds of logics to support inference; description logic, first order logic, temporal logic and spatial logic to name a few. Moreover, many kinds of reasoning over uncertainty such as Bayesian inference or fuzzy logic can also be applied.

The reasoning service is used by some context mapping services and context aggregators. They invoke the reasoners through a fixed API, providing the reasoners with a context data which can be considered as a knowledge base containing all the facts needed for inference. All new inferred facts will be inserted into that context data for later queries. The use of a fixed interface for all kinds of reasoning engine makes it possible to add and handle different reasoners. The developers can then use any kind of reasoning they want.

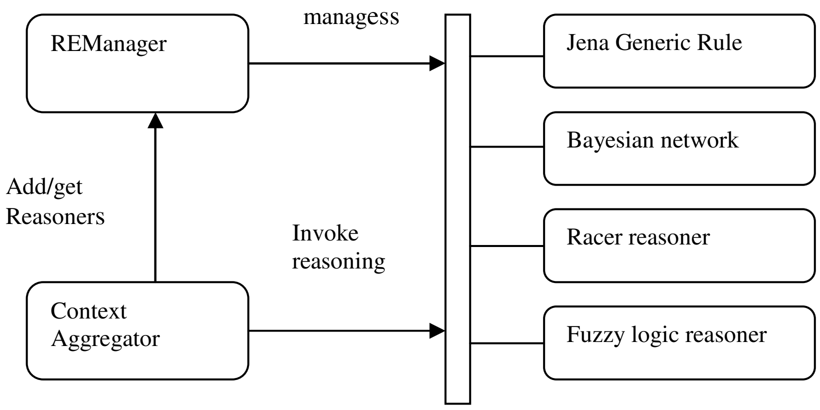

Fig. 1. Reasoning engine in our middleware

3.2 Main Characters of Our Reasoning Engine

(1) **Multiple Reasoning Mechanism.** Considering different upper applications' different reasoning requirements, multiple reasoning mechanisms are proposed. Agents can reason about context using rules written in different types of logic like first order logic, temporal logic, description logic (DL), higher order logic, fuzzy logic, etc. Different agents have different logic requirements. Agents that are concerned with the temporal sequence in which various events occur would need to use some form of temporal logic to express the rules. Agents that need to express generic conditions using existential or universal quantifiers would need to use some form of first order logic (FOL). Agents that need more expressive power (like characterizing the transitive closure of relations) would need higher order logics. Agents that deal with specifying terminological hierarchies may need description logic. Agents that need to handle uncertainties may require some form of fuzzy logic.

Instead of using rules written in some form of logic to reason about context, agents can also use various machine learning techniques to deal with context. Learning techniques that can be used include Bayesian learning, neural networks, reinforcement learning, etc. Depending on the kind of concept to be learned, different learning mechanisms can be used. If an agent wants to learn the appropriate action to perform in different states in an online, interactive manner, it could use reinforcement learning or neural networks. If an agent wants to learn the conditional probabilities of different events, Bayesian learning is appropriate.

(2) **Pluggable Reasoning Modules.** To provide more help to developers so that they can concentrate on developing rules or networks for reasoning and not be burdened with the low-level details, our middleware defines wrappers for each reasoner type. For example, a wrapper of Jena generic rule reasoner allows the developer to easily add a new reasoner just by declaring the rule file name and some namespace abbreviations. The following piece of code illustrates how to add and invoke a rule-based reasoner.

```
  /* add a new reasoner providing the rule file*/
 ContextReasonerManager.addReasoner("Location",ReasonerType.GENERIC_REASONER,
"etc/contel.rules") ;
  /*declare some statements*/
  sms=new ContextStatement[ ] {PastLocationDescription, hasLocation};
  /*invoke the reasoner to do reasoning, providing the reasoner name, the context data name and the re-
  quired statements*/
  cdm.invokeReasoning("Location", "Data", sms);
```

(3) **Ontology Reasoning Mechanisms.** Ontology reasoning helps us to find subsumption relationships (between subconcept-superconcept), instance relationships (an individual i is an instance of concept C), and consistency of context knowledge base, provided by Racer Server. In the design phase of formalizing the context entities, OWL reasoning services (such as satisfiability and subsumption) can test whether concepts are non-contradictory and can derive implied relations between concepts.

Let us take an example to see how ontology reasoning can help deducing implied context. In location ontology, the property locatedIn is a TransitiveProperty, and isPartOf is subProperty of locatedIn. So when knowing that Bilbo is locatedIn Bed, and Bed is a part of BedRoom which is part of Home, the system can deduce that Bilbo is locatedIn BedRoom and Home.

4 Improvement on Our Reasoning Engine

In this section, we will present our idea on how to improve our existing reasoning engine described in last section.

Knowledge base is an indispensable part of our reasoning engine. For rule-based reasoning, the knowledge base provides many user-made rules. As for machine learning reasoning, many learning models are required to be loaded into the knowledge base. These learning models are deduced by analyzing context history using machine learning methods. Since both rule-based and machine learning reasoning are supported by our reasoning engine, the knowledge base of our system includes two parts: user-defined rules and learning models. The process to construct a knowledge base is shown in Fig. 2.

Knowledge base plays an important role in knowledge-based reasoning systems. The content quality in knowledge base directly influences the reasoning result. So improving the quality of rules and models in our knowledge base is urgently needed. Currently, there are some potential problems with our knowledge base.

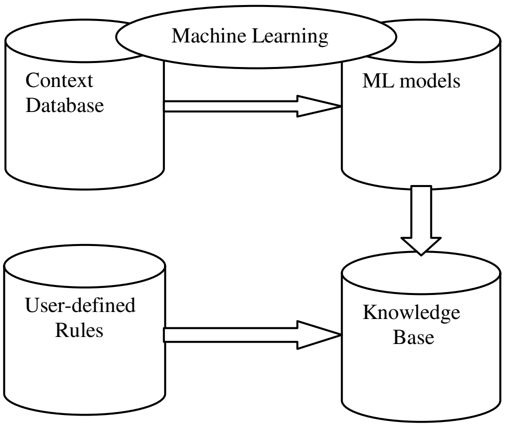

Fig. 2. Knowledge base of our reasoning engine

4.1 User-Defined Rules in Knowledge Base

Rule-based reasoning has proved itself effective decision makers for many types of problems. However, the accuracy of such systems is highly dependent upon the accuracy of the user's domain theory. When users learn or create a set of rules, they are subject to a number of hindrances. As a result, the user-defined rules are always incomplete or erroneous. In ubiquitous environment, the rules for high-level context reasoning usually follow this format: if $lc_1 = x_1$ and/or $lc_2 = x_2$ ….and/or $lc_n = x_n$, then $hc = M$, in which lc_i (i=1,2,...n) represents low-level context and hc represents high-level context. It is easy to see that hc is determined by the value of lc_i . In ubiquitous environment, the number of different types of low-level context is huge. In some simple cases, it is feasible for users to manually choose which low-level context should be used to deduce a given high-level context. However, as for some complex high-level context reasoning, such as activity and mood reasoning, users are not able to exactly determine which low-level context should be selected. As a result, many useless low-level contexts are mistakenly selected to deduce a given high-level context, which makes the rules used for some high-level context reasoning is not correct.

To solve the above problem, rules refinement is urgently needed. In this paper, we propose to use feature selection method to analyze context history, so that those low-level context, which has more weight to deduce a give high-level context, could be figured out. Then, these extracted low-level contexts could be used to compare with those provided by users to improve user-defined rules' quality.

4.2 Learning Models

We got learning models by utilizing machine learning methods to analyze context history. Compared with user-defined rules, the accuracy of learning models is improved. However, one potential problem still exits.

Here, we take user activity as an example of high-level context. In general, activity could be divided into several classes. Hence, this high-level context deduction could be treated as a classification problem.

Classification systems depend upon having the best set of input features from which a classification decision can then be made. This is true both for the classifiers themselves and for the learning models might be used to classify. This drives us to select the "relevant" low-level contexts for training, instead of all the low-level contexts.

To achieve this function, firstly, we still utilize feature selection method to find the most relevant low-level context for a given high-level context. Then, different inductive learning methods could be used to deduce the learning models. Many work [17][18] showed that the learning models deduced from relevant features (low-level context) could be much improved. Now the process to build a knowledge base for reasoning engine can be depicted by Fig. 3.

In addition to refine rules and learning models, other advantages using feature selection method include:

1) Saving sensors. Since useless low-level contexts could be detected by using feature selection method. So in real ubiquitous environment, only those useful sensors need to be deployed. As a result, the cost to build a smart environment is reduced.
2) Reducing human's burden. Now many experiments are based on the sensors attached to human. Human's burden could be reduced by filtering those useless sensors.
3) Saving context database size. Most algorithms in reasoning engine are supervised learning method, which need to analyze historical context data to design an accordingly model. In a ubiquitous environment, effectively mining useful low-level context can avoid data explosion.

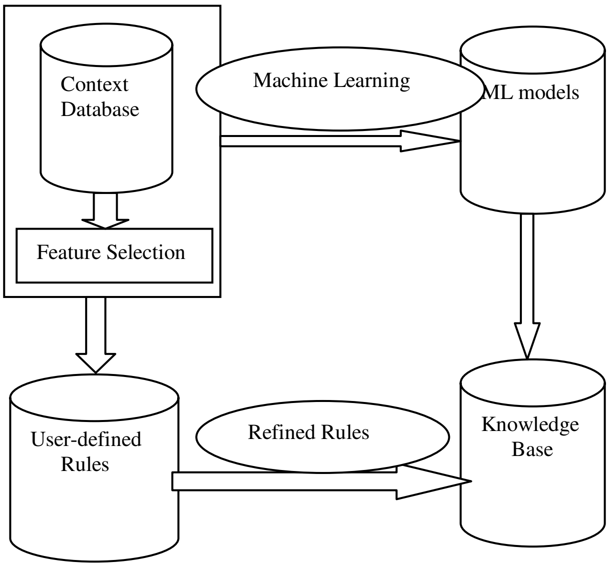

Fig. 3. Knowledge base construction based on feature selection

4) Easy to see the relationship between low-level context and high-level context. Since most irrelevant features have been filtered by feature selection, it is easy to make human-understandable rules.
5) Reducing reasoning uncertainty. As we all know that sensors have their inherent uncertainty. The more sensors used for reasoning, the more uncertainty will arise.

5 Conclusions

In this paper, we propose to utilize feature selection method in our context-aware middleware reasoning engine. Our reasoning engine supports both rule-based reasoning and machine learning reasoning. The main advantages using this engine are described in our paper.

Feature selection method is used to filter those low-level contexts which do not contribute a lot for a given high-level context reasoning. The direct effect is that fewer sensors are required. As a result, context database occupation is reduced.

Our plan for future work is to assess our reasoning engine in our smart office. We are currently deploying different sensors in our office and will use them for collecting empirical data. Finally more high-level contexts will be inferred to further prove the feasibility of our reasoning engine.

Acknowledgement

This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) Support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2006-(C1090-0602-0002)).

References

1. Dey, A.K., Abowd, G.D., Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. J. of Human-Computer Interaction (HCI) 16, 97–166 (2001)
2. Hong, J.I., Landay, J.A.: An Infrastructure Approach to Context-Aware Computing. J. Human-Computer Interaction (HCI), London, UK, 287–303 (2001)
3. Shafer, S.A.N., Brumitt, B., Cadiz,J.J.: Interaction Issues in Context-Aware Interactive Environments. J. Human-Computer Interaction (HCI), London, UK, 363–378 (2001)
4. Pascoe, J., et al.: Issues in Developing Context-Aware Computing. In: Proc. of the International Symposium on Handheld and Ubiquitous Computing, pp. 208–221. Springer, Heidelberg (1999)
5. Schilit, W.N.: A Context-Aware System Architecture for Mobile Distributed Computing. PhD Thesis. Columbia University (1995)
6. Hung, N.Q., Shehzad, A., Kiani, S.L., Riaz, M., Lee, S.: A Unified Middleware Framework for Context Aware Ubiquitous Computing. In: The 2004 IFIP International Conference on Embedded And Ubiquitous Computing. EUC2004, Japan, pp. 672–681 (2004)
7. Dey, A.K.: Understanding and Using Context. Personal and Ubiquitous Computing 4–7 (2001)
8. Kulkarni, A.: A Reactive Behavioral System for the Intelligent Room. Master's Thesis in Computer Science and Engineering at the Massachusetts Institute of Technology. Cambridge, MA (2002)
9. Brumitt, B.L., Meyers, B., Krumm, J.: EasyLiving: Technologies for Intelligent Environments. In: Proc. of Handheld and Ubiquitous Computing, pp. 12–27 (2000)
10. Elrod, S., Hall, G., Costanza, R., Dixon, M., Des Rivieres, J.: Responsive Office Environments. Proc. of ACM Communications , 84–85 (1993)
11. Gaia Project (2005), http://gaia.cu.uiuc.edu/
12. Gu, T., Pung, H.K., Zhang, D.Q.: A middleware for building context-aware mobile services. In: Proc. of IEEE Vehicular Technology Conference (VTC 2004), Italy, pp. 2656–2660. IEEE Computer Society Press, Los Alamitos (2004)
13. Jena: A Semantic Web Framework for Java, http://jena.sourceforge.net/
14. Fahy, P., Clarke, S.: CASS – a middleware for mobile context-aware applications. In: Proc. of the Workshop on Context Awareness, MobiSys (2004)
15. Chen, H., Finin, T., Joshi, A.: An ontology for context-aware pervasive computing environments. In: Proceedings of the Workshop on Ontologies in Agent Systems, pp. 197–207 (2003)
16. Ipina, D., Katsiri, E.: An ECA Rule-Matching Service for Simpler Development of Reactive Applications. In: Proc. of Middleware 2001 at IEEE Distributed Systems Online 2(7) (2001)
17. Liu, H., Yu, L.: Toward Integrating Feature Selection Algorithms for Classification and Clustering. IEEE Transactions on Knowledge and Data Engineering 1–12 (2005)
18. Kwak, N., Choi, C.-H.: Input Feature Selection for Classification Problems. IEEE Transactions on Neural Networks, 143–159 (2002)