Middleware Architecture for Context Knowledge Discovery in Ubiquitous Computing

Kim Anh Ngoc Pham, Young Koo Lee, and Sung Young Lee

Department of Computer Engineering, Kyung Hee University, Korea
anhpnk@oslab.khu.ac.kr, yklee@khu.ac.kr, sylee@oslab.khu.ac.kr

Abstract. Advanced analysis of data for extracting useful knowledge is the next natural step in the world of ubiquitous computing. So far, most of the ubiquitous systems process knowledge in problem-specific or domain-specific manners. This article introduces the concept of context knowledge discovery process, and presents a middleware architecture which eases the task of ubiquitous computing developers, while supporting data mining and machine learning techniques. We show how the middleware architecture supports building ubiquitous systems which are able to "learn" and "think" by introducing some learning – reasoning combination mechanisms, such as the context recognition and prediction, or the deductive rule learning and reasoning process. [1]

1 Introduction

Context-aware ubiquitous computing emphasizes on using context of users, devices, etc. to provide services appropriate to particular person, space, and time. Every computing system dealing with the user should take into account human behavior in one way or the other to materialize ubiquitous computing experiences for him. As we all know that 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 [1].

A lot of work has been done in the area of context-aware computing, in which most of them are only concerned with one or more aspects in an ad hoc manner. Context Toolkit [2] uses the concept of widget to obtain raw contextual information from sensors and passes them either to interpreters or to servers for aggregation. Interpreters and servers use simple HTTP protocol for communication and the XML (name-value pairs only) as the language model for the context. In [3], graph based model for context aggregation and dissemination is proposed where contextual information sources are modeled as event publishers, while context-aware applications as event subscribers. Context Fabric [1] provides a distributed context-aware infrastructure to support the acquisition and retrieval of context data using an entity-relation style logical context data model, encoding data in XML and utilizing

[1] This work was supported by MIC Korea. Dr. S.Y.Lee is the corresponding author.

L.T. Yang et al. (Eds.): EUC 2005, LNCS 3824, pp. 1177 – 1188, 2005.

XPath as the query language. Gaia [4] is also a distributed middleware infrastructure supporting context-aware agents in smart spaces. It adopted a predicate model of context data encoded in DAML ontologies, and proposed that different logic reasoning and machine learning techniques can be adopted to support context inference.

Based on our knowledge, we have come up with a set of key issues [5]; a context-aware ubiquitous computing system should tackle in order to successfully deploy in real life, namely unified sensing framework, formalized modeling and representation, supporting multiple reasoning approaches, and finally delivering the context to applications on semantic matchmaking basis. And the heart of those intelligent systems is the reasoning and learning capability. Consider a Smart Home scenario, when a user enters his home, the system should be able to recognize who the user is, what he is doing, "guess" what he intends to do, and how he desires the system to assist him. A good reasoning engine can help the system reason properly, but how to build that good reasoning engine turns out to be a difficult question.

Many current Ubiquitous systems are using reasoning engines to infer high-level information from the low-level context data. However, the performance is limited due to the complications in composing rules for the rule-based reasoners, or calculating the uncertainty in probabilistic reasoners. If we can extract some useful rules from the database e.g. "If Bilbo is on bed and it is Bilbo's sleeping time then the probability of Bilbo being sleeping is 85%", then those rules can be used to automatically construct rule-based reasoners. Hence knowledge discovery and data mining come into play. Knowledge discovery and data mining (KDD) deal with the problem of extracting interesting associations, classifiers, clusters, and other patterns from data [10]. KDD is playing an increasingly important role in business, scientific, and engineering applications. In ubiquitous computing environments where large quantity of data is available, advanced analysis of data for extracting useful knowledge becomes the next natural step.

In this paper we propose a middleware architecture for discovering the valuable knowledge from large and heterogeneous amount of context data. We have built and deployed context knowledge discovery architecture as a central part of CAMUS [5] - a middleware infrastructure for context-aware ubiquitous systems.

The rest of this paper is organized as follows: after a brief explanation of context knowledge discovery process in section 2, we present the architecture of CAMUS – our Context-Aware Middleware for Ubiquitous Systems in section 3. In section 4, we give a detailed description of our idea of applying data mining into ubiquitous computing by discussing the rule learning and context recognition mechanism. We conclude our paper with a summary and outlook in section 5.

2 Context Knowledge Discovery

2.1 What is Context Data

Context has been defined by Dey [6] as *any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or a*

physical or computational object which we adopt in this article. A good overview on different definitions of context can be found in [7].

The amount of context data maintained in the system is huge, due to the large number of data sources and from the context history. Since the main task of Data Mining is to extract valuable knowledge from a large amount of data, it is a fundamental requirement for applying data mining in ubiquitous system.

What can we mine from context data? Currently in CAMUS we mine the deductive rules which are used to build rule-based reasoning engines (those reasoning engines then will be used to infer high-level context data from low-level context). We also use clustering technique to discretize the time value and find out the common activities of users. While other systems label the activities in advance, in our system we identify activities by finding clusters of common place, time, environment parameters, and device status. Next section shows the context data mining process, which are somehow different from "traditional" transactional data mining.

2.2 Context Knowledge Discovery Process

Fig. 1 illustrates the context knowledge discovery process and its position in the ubiquitous data inquiry and knowledge management process.

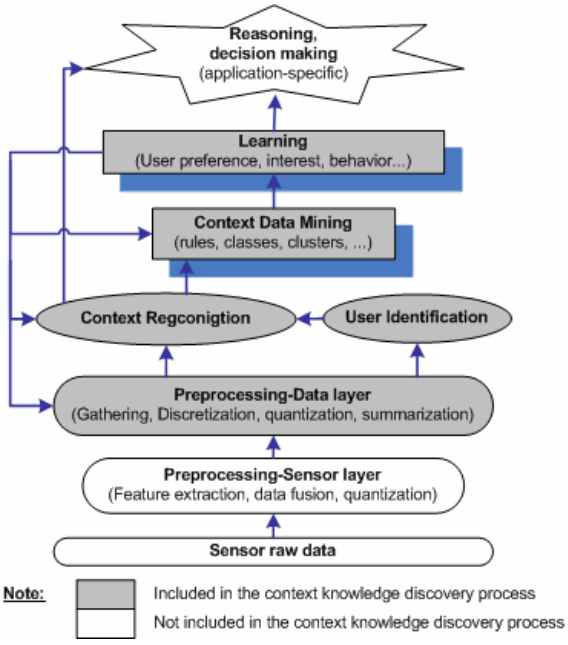

Fig. 1. Context knowledge discovery process and its position in the ubiquitous data inquiry and knowledge management process

Data inquiry and preprocessing at sensor layer: Low-level context data acquired from various sensors is preprocessed by the Feature Extraction Agents (described in

section 3) before sending into Context Repository. This preprocessing step at sensor layer includes feature extraction, data fusion, quantization and encapsulation of sensing data. For example, 3 wireless LAN access points deployed in one area can act as location sensors by extracting and fusing the signal strength. At this step, some features are further quantized into nominal values, for instance the light intensity can have 5 levels of quantized value: dark, dim, normal, bright, very bright.

The extracted features from sensor layer are then forwarded to context data mining process including 4 main steps:

i) Context data preprocessing *(in data layer)*
One important task of this step is gathering relevant context data into data cubes. A context data cube may have multi dimensions: time, location, user, environment parameters (light intensity, sound level, temperature, etc.), device status (television, air conditioner, door status, computer running programs, etc.). The context data has to be user centric; i.e. the environment parameters and device statuses are of the environment where user is currently located in. In order to have a uniform representation of the disparate features, they are further summarized, quantized or discretized, and normalized.

When mining the user preferences or user behavioral habits, user commands to control the devices and appliances are aggregated. Current or scheduled activities are also included into the data cube if available.

ii) User identification and context recognition
Recently, user identification has become an easy task with the advance of identification techniques such as RFID tag, or just simply base on devices associated with the user such as PDA, mobile phones, computer logging on, and the like..

Meanwhile, context recognition remains difficult. Many systems are working on context recognition following different approaches such as using neural network, Bayesian network, etc [17]. Each approach has its pros and cons. In the context data mining process of CAMUS, we use clustering and classification techniques to recognize the context (current activity) of user. The detail of these techniques will be discussed in section 4.

iii) Context data mining
Using data mining techniques to mine context data, this module produces the association rules, classification rule sets and clusters. Output of this module is the input of learning module.

iv) Learning module
This module is an interface between data mining module and the rest of context discovery process. It evaluates the rules and decides which one can be used for reasoning, parses the clusters into classes, maps the rules into the right format (the rule format of the reasoning engine; for example, if we use Jena generic rule reasoners, the rules must have Jena rule format [16]). It also works as a manager to request for new rules, manage the rules and rule mining process, insert or update rule sets of reasoning engines, and so on. The learning module output is used to support all other context data discovery steps: to preprocess data, recognize user context, or

supervise the context data mining. The function of this module will be illustrated by rule learning mechanism in section 4.

Our CAMUS middleware architecture provides maximum support for this context data mining process.

3 CAMUS - Middleware Architecture for Ubiquitous Systems

The knowledge discovery process presented in this paper is the crucial part of our CAMUS architecture, a unified middleware framework for context-aware ubiquitous computing. Here we briefly describe the core functional components of CAMUS as depicted in Fig.2, more details can be found in [5].

3.1 Feature Extraction Agents

Feature Extraction Agents (FX Agents), or wrappers of sensors, extract the most descriptive features for deducing contexts in upper layers, sometimes attached with their semantic meanings and uncertainties. Then, *Feature - Context mapping layer* will perform the mapping required to convert a given feature into elementary context. This module handles the sensor layer pre-processing step in CKDD.

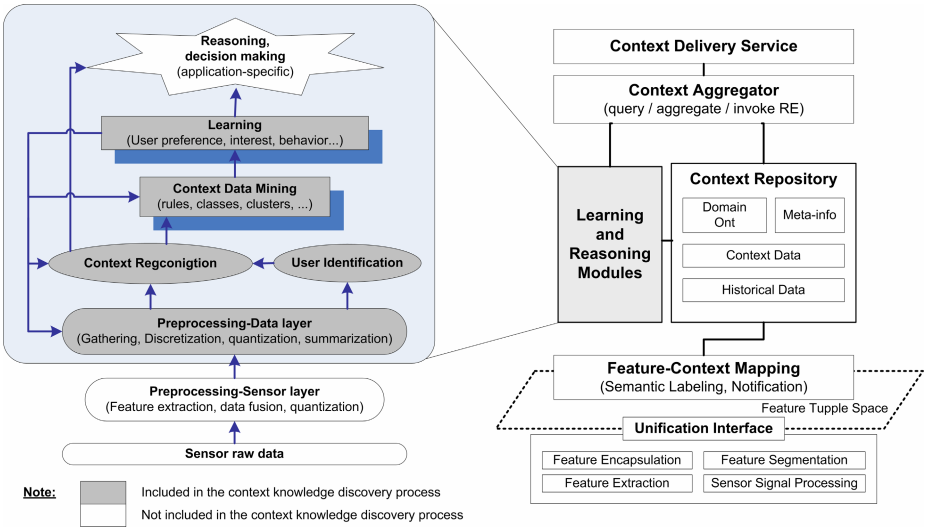

Fig. 2. The correlation between CAMUS architecture and CKDD process

3.2 Knowledge Processing Layer

This module handles the main steps of CKDD process.

Context Repository provides the basic storage services in a scalable and reliable fashion and contains the domain ontology (concepts and properties), contextual

information (including both elementary and composite contexts), and meta-information such as the meta-information about the input, output and capabilities of pluggable reasoning modules.

The Context Repository consists of two databases: the main database which contains all current context data, and the history database. Whenever a new context data comes, the corresponding old context data will be moved into history database. In this way, the size of main database is reduced; consequently it maintains good performance in context query processing.

Reasoning Engine is a collection of various pluggable reasoning modules to handle the facts present in the repository as well as to produce composite contexts. Since not all information can be gathered from sensors, and sometimes the most interesting kinds of context are those that humans do not explicitly provide, context reasoning mechanisms are needed. For instance, the current activity of a user could be inferred based on a combination of many other contexts, e.g. his location, his gestures, time of the day, environment status, etc [5]. Each reasoning mechanism has its own expressiveness, for example Description Logics (DL) is suitable for specifying terminological hierarchies while Spatio-temporal Logic is suitable concerning spatial-temporal sequence in which various events occur, and Bayesian Networks [8] are appropriate for learning the conditional probabilities of different events. Thus a middleware infrastructure needs to provide support for incorporating different reasoning mechanisms into the system, as well as specifying the appropriate mechanism for each context. This will facilitate not only the system internal modules to infer high-level context from low-level or predefined context, but also the applications to reason for their own application-specific context.

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 infrastructure 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 required statements */
cdm.invokeReasoning("Location", "Data", sms);
```

To help constructing the reasoning engine, *Machine Learning Modules* utilize many machine learning and data mining techniques to learn the logic, model (such as user preference model), or train the Bayesian networks and neural networks, using historical context data as training data.

3.3 Context Delivery and Aggregator Services

In our middleware framework, each *context aggregator* (analogous to web service) specifies the context it provides, by utilizing the concepts defined in the ontology repository. This standard schema sharing allows different kinds of entities to be described and utilized by delivery service to find useful services needed by the applications, thus, allowing a flexible mechanism for exchanging descriptive information of various entities.

Context delivery provides the services (context aggregators) with the registration interface to make their information known to the applications. The matchmaking module matches the appropriate service with the client provided the access control policies are not violated.

4 Application of Data Mining Techniques in Ubiquitous System

4.1 Learning the Rules for Context Reasoning

Rule based reasoning is currently the most common form of knowledge processing. Especially in the field of context-awareness, many systems are using semantic knowledge base which fully supports rule-based reasoning engines such as in [4]. However, most developers find building the rules the most difficult task in building ubiquitous computing systems.

In order to minimize the burden for developer in building context-aware applications, our middleware architecture provide support to learn the inference rules from context data and build reasoning engines using those rules, as depicted in Fig. 3.

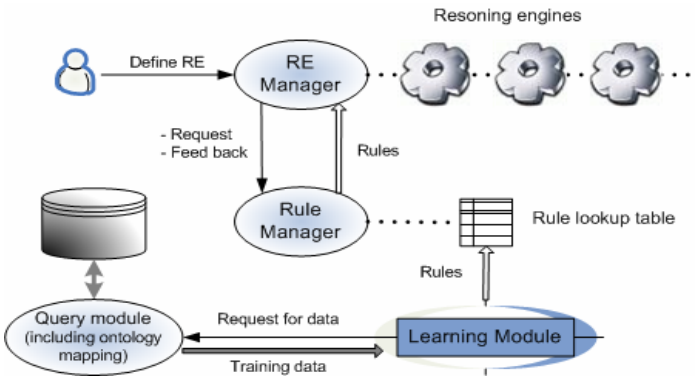

Fig. 3. Rule learning mechanism in CAMUS

To explain the rule learning mechanism in detail, we will first describe a simplified Smart home scenario (only relates to user preference) and how a developer can make use of the rule learning architecture of CAMUS to build a rule-based reasoner which can infer the user preference from context data.

4.1.1 The Simplified Smart Home Scenario

In a Smart Home environment, computing devices are spread throughout the environment, present everywhere in the house to provide services and convenience to the people living inside. The context of user and the status of home environment are sensed by various kinds of sensors, e.g. light sensor, temperature sensor, fire sensor, etc. There are also a set of software agents which control the home devices and provide services to the user seamlessly and invisibly without any explicit user intervention. The following use case is designed focusing on how the Smart Home system adapts to user desire:

Bilbo is sleeping in his bed room an early morning. Since he prefers darkness while sleeping, the system maintains a Dark light intensity level by automatically dimming down the light and rolling down the curtain when the outside becomes brighter. Because normally Bilbo wakes up at around 7:30, at that time the system decides to wake him up and fire the alarm clock.

When Bilbo finishes washing and enters dining room, the television is turned on. Recently he likes to watch the Sport News in the morning, so the system automatically selects the channel which is providing Sport News.

4.1.2 Learning User Preference as Rules

In this scenario, the user preference is learned through user control commands and responses to the messages from system. User can control the home devices by remote controls, or send command messages through some computer software interfaces. The commands are stored in the historical database together with relevant information i.e. user location, timestamp, current activity, environment state. The tuple {user=Bilbo; place=DiningRoom; timestamp = 2005/04/01 8:40; currentActivity=WatchingTV; command= TV.SelectChannel; parameter=SportNews} is an example.

Using this kind of information as the training data, the system can generate rules like:

personName=Bilbo; currenttime=[8:40-8:50]; currentActivity=WatchingTV
⇒ command=TV. SelectChannel; parameter=SportNews (Utility=0.86)

The rules can then be converted into suitable format for the reasoning engine which are used. In our prototype system, because we use Jena generic rule reasoner to reason over OWL[15] format data, the final rule has Jena rule format[16]:

```
[r1:    (?user agt:personName "Bilbo"), (?x time:currentTime ?t),
  (?user act:currentActivity ?curact), (?curact rdf:type act:WatchingTV)
  AND 520 <=?t AND ?t <= 530
  --> [(?pref agt:hasValue "TV.SelectChannel"), (?pref agt:hasPara "SportNews")
        <-- (makeInstance(?user,agt:hasPreference, agt:Command, ?pref) ]]
```

with agt, time, act, rdf, act are the aliases for the namespaces of ontologies which are currently used to define to data model of our context aware system.

4.1.3 Initiate Rule Learning Process

Because the purpose of rule learning is to supply rule sets for reasoning engines, we first take a look at the information needed to define a rule-based reasoner. Even when the rule set will be learn automatically, the developer must figure out the *set of data* on which the reasoner will do its reasoning. For example, in this smart home scenario, user preference is learn based on user, time, location, environment parameters, user control commands, user activities. Therefore the developer should first define the set of concepts which will be included in the data supplied to the future reasoning engine.

The developer can also define desired *outputs* of the desired reasoning engines. However, the indication of output is optional, because in some cases we don't know exactly the desired output. For example, from activity and other information, we can infer the intended control commands, but also we can base on the user control commands to guess the current activity. In the above scenario, the desired output is user preference.

The Reasoning Engine Manager then sends request to Rule Manager with the required data and/or output of the rules. The learning module will send data request to the query module (part of Data layer Preprocessing module) to get the data needed for mining such rules. Query module responses to learning module with the data cubes.

4.1.4 Rule Mining Algorithm

In ubiquitous systems, a rule mining algorithm has to fulfill following requirements: online (learning has to be done continuously in time domain), adaptive (as user behaviors change over time, rule sets must be adapt to new input data), rules have high accuracy but not necessarily high coverage (For example, every Sunday the user will see a special program on television; the coverage of the control command to select this program is low, but the related rule is highly accurate).

If the output is defined, we use the Sequential-Covering algorithm [9] to learn the first rule sets from example data set. Otherwise, Apriori association rule mining algorithm [10] is more suitable. After learning one rule, the examples covered by the rule will be removed, and then the learning process repeats until no positive example is left.

At this step, a question is raised: How can we select the "right" rules among a large number of learned rules? To address this problem, we assign a utility function to calculate the value of each rule based on confidence and support.

$$Utility(r) = \alpha.Conf(r) + \beta.Sup(r) \quad (1)$$

With *Conf(r)* is the confidence *Sup(r)* is the support of the rule.

The α and β coefficients are related to each other by $\alpha + \beta = 1$, and define the type of rule which is more interested. Normally $\alpha = 1$ and $\beta = 0$, showing that a rule which has high confidence will be chosen even if it rarely happens.

Among a large number of rules mined, only the rules with high utility will be selected. Those rules are inserted into rule table, and supplied to Reasoning Engine Manager to be added into the rule set of the reasoning engine.

Whenever a reasoning engine is invoked, some rules will be fired. The utility of those fired rules can be increased or decreased depending on the feedback from the application after receiving new inferred information is positive or negative. In case of Smart Home system, there are two common types of rule: decision rules and

prediction rules. Decision rule leads to a control command decision, such as turn on the light or adjust the temperature. After the control command is executed, there are two types of response from user:

- If the user satisfies with the automatic command, he will do nothing.
- If the user does not satisfy, he himself will give a control command to adjust the environment as to his desire.

In case of prediction rules, by observing the user, the system can know whether the prediction is right or wrong.

According to the feedback from application, the rules in rule set can be divided into 3 groups: the "right" rules (which generate right prediction or right commands), the "wrong" rules (which generate wrong prediction or unsatisfying commands) and the rules which are not fired.

The utility of those rules then be updated:

$$Utility(r_r) = Utility(r_r) + \gamma \times (1 - Utility(r_r) \tag{2}$$

$$Utility(r_w) = Utility(r_w) - \gamma \times Utility(r_w) \tag{3}$$

$$Utility(r_u) = Utility(r_u) - \varepsilon \times Utility(r_u) \tag{4}$$

Where r_r denote the "right" rule, r_w denotes the "wrong" rule and r_u denotes the unfired rule. γ and ε are the coefficients to tune the speed of changing the utility of rules after each time the reasoning engines are invoked. If ε is set to 0, the utility of unfired rules is maintained.

The rules which have utility below a threshold will be removed from reasoning engine's rule set.

User preference and behavior changes quickly, so the new data plays more important role in the training data set than the old data. To maintain a high learning performance, we can restrict the number of training data set. Only N newest history data records will be retrieved. In that case, whenever new context info comes into the history database and makes the training data set change, the utility of the rules in rule table is updated:

$$Utility(r) = \frac{Utility_{old}(r) \times N + Utility_{new}(r)}{N + 1} \tag{5}$$

Where $Utility_{old}(r)$ is the current utility of the rule and $Utility_{new}(r)$ is the new utility of the rule calculated by the new training data.

Then the examples covered current rules are removed and new rules are mined.

4.2 Context Recognition Using Clustering

In section 2, we briefly described the context recognition step. This section discusses it in detail.

The task of this step is to identify the context (or activity, situation, etc.) of user, based on the low-level contexts from sensors. After preprocessing, the sensor data is gathered in context data cube, including user, time, location, environment parameters and device statuses. To know the context of user, this step handles two tasks:

determining the common contexts of user using clustering over history context data and identifying the current context of user based on sensed data.

4.2.1 Clustering Common User Contexts

Ideally, an algorithm for context recognition must be online and thus unsupervised and must have a variable network topology to cope with changing sensor data cube (changing sensor configurations). The clustering algorithm must not be hard competitive to allow multiple active contexts and it has to be designed for life long learning to not forget or overwrite already learned clusters over time (which is known as the plasticity-stability dilemma [11] in neural networks and clustering literature). Among the most common online clustering algorithms, LLGNG [12] – a modification of GNG algorithm [13] to cope with continuously changing environments and life-long learning - seems to provide most flexibility for context clustering in ubiquitous environment. After the clustering step, the system can label the clusters by itself, or ask user to give some meaningful names, such as "working", "watching TV", "cooking" and the like for the clusters. This is theoretically an optional step, because the machine considers the meaningful and non-meaningful labels the same. However, in practice, while naming the clusters, user can give same name for many clusters, or split a cluster into many different contexts; and hence improve the result of context recognition.

4.2.2 Identify Current Context

This is not the task of data mining but machine learning. Having the result clusters as the context classes, we can apply different machine learning methods to infer the current context/activity/situation of user, base on the time, location, environment and other low-level context data.

Selecting the learning mechanism is up to developer. Our middleware provides the possibility of plugging in different reasoning engines and learning modules. We also provide advance support for the common reasoning engines, such as rule-based reasoning engines (as described above). Another supported approach is Bayesian network, which gains an advantage in the uncertain environment.

CAMUS support Bayesian network by: First, associating each context with a probability attribute; Second, enhancing the context data ontology (implemented in OWL) to represent some concepts which support creating and maintaining Bayesian network, such as relation chain, probabilistic dependencies, etc. [14]; Finally, introducing a Construct-BN algorithm [14] to derive a Bayesian network from the domain ontology.

5 Summary and Outlook

In this paper, we have presented a novel middleware architecture which addresses the key challenges in ubiquitous computing systems, including supports for a context knowledge discovery process. Our CAMUS prototype proves the feasibility of building intelligent ubiquitous environments with lesser workload on the developers.

Our next steps include the effort to make the interface between learning and reasoning more transparent and make more use of data mining techniques.

References

[1] Hong, J. I., et al.: An Infrastructure Approach to Context -Aware Computing. HCI Journal, 2001, Vol. 16.
[2] Context Toolkit project http://www.cs.berkeley.edu/~dey/-context.html
[3] Guanling Chen and David Kotz.: Solar: An Open Platform for Context –Aware Mobile Applications. In Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002), Switzerland, June, 2002.
[4] Anand Ranganathan, Roy H. Campbell: A Middleware for Context -Aware Agents in Ubiquitous Computing Environments. In ACM/IFIP/USENIX International Middleware Conference, Brazil, June, 2003.
[5] Anjum Shehzad et. al. A Comprehensive Middleware Architecture for Context-Aware Ubiquitous Computing Systems. Accepted for publication, ICIS 05.
[6] A. Dey, G. D. Abowd, and D. Salber. A context-based infrastructure for smart environments, 1999
[7] A. Schmidt. Ubiquitous Computing – Computing in Context. PhD thesis, Lancaster University, November 2002.
[8] Korpipaa, P., Koskinen, M., Peltola, J., Makela, S. M., Seppanen, T.: Bayesian approach to sensor-based context awareness. In: Personal and Ubiquitous Computing, Vol. 7, Issue 2. (July 2003) 113-124
[9] Ton Weijters Jan Paredis. Discovering Rules with a Genetic Sequential Covering Algorithm (GeSeCo)
[10] Jiawei Han,Micheline Kamber. Data Mining: Concepts and Techniques
[11] S. Grossberg. Adaptive pattern classification and universal recoding: Parallel development and coding of neural feature detectors. Biological Cybernetics, 23:121–134, 1976.
[12] F. H. Hamker. Life-long learning cell structures continuously learning without catastrophic interference. Neural Networks, 14(4–5):551–573, May 2001.
[13] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, Advances in Neural Information Processing Systems 7, pages 625–632. MIT Press, Cambridge MA, 1995.
[14] Binh An Truong. Modeling and Reasoning about uncertainty in Context-Aware Systems. Accepted for publication, ICIS 05..
[15] W3C Web Ontology Working Group: The Web Ontology language: OWL. http://www.w3.org/2001/sw/WebOnt/
[16] Jena: A Semantic Web Framework for Java. http://jena.sourceforge.net/
[17] Mozer, M. C. (2005), Lessons from an adaptive house. In D. Cook & R. Das (Eds.), Smart environments: Technologies, protocols, and applications (pp. 273-294). Hoboken, NJ: J. Wiley & Sons