Cabot: On the Ontology for the Middleware Support of Context-Aware Pervasive Applications

Chang Xu[1], S.C. Cheung[1], Cindy Lo[1], K.C. Leung[1] and Jun Wei[2]

[1]Department of Computer Science, Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
[1]{changxu, scc, cindylo, lkchiu}@cs.ust.hk

[2]Technology Center of Software Engineering, Institute of Software, Chinese Academy of Science
[2]wj@otcaix.iscas.ac.cn

Abstract. Middleware support is a major topic in pervasive computing. Existing studies mainly address the issues in the organization of and the collaboration amongst devices and services, but pay little attention to the design support of context-aware pervasive applications. Most of these applications are required to be adaptable to dynamic environments and self-managed. However, most context-aware pervasive applications nowadays have to carry out tedious tasks of gathering, classifying and processing messy context information due to lack of the necessary middleware support. To address this problem, we propose a novel approach based on ontology technology, and apply it in our *Cabot* project. Our approach defines a context ontology catered for the pervasive computing environment. The ontology acts as the context information agreement amongst all computing components to support applications with flexible context gathering and classifying capabilities. This allows a domain ontology database to be constructed for storing the semantics relationship of concepts used in the pervasive computing environment. The ontology database supports applications with rich context processing capabilities. With the aid of ontology technology, *Cabot* further helps alleviate the impact of the naming problem, and support advanced user space switching. A case study is given to show how *Cabot* assists developers in designing context-aware pervasive applications.

1 Introduction

A pervasive computing environment encompasses a spectrum of computation and communication devices that seamlessly augment human thoughts and activities [1]. Due to the non-trivial context management inherent in pervasive computing, a suitable software infrastructure is needed to assist the development of context-aware pervasive applications. We refer the **context** of a computation task to as the circumstances or situations in which the task takes place. Most context-aware pervasive applications are required to be adaptable to highly dynamic environments and self-managed. Therefore, the design of such applications is a challenging research issue.

At present, developers of context-aware pervasive applications need to write tedious and repetitive codes to handle context management, which concerns the following three functions:

- **Context gathering:** Gather proper context information from relevant context sources in a flexible way rather than specifying them explicitly. When an application is interested in object movement, the middleware should be able to select proper sensors to collect context information about object movement.
- **Context classifying:** Classify context information into different categories in an application-specific way. An application may hope to analyze a certain scenario where the subject is "human being", the action is "enter" and the area is "office 4208". The common context classification is only based on context type (e.g. sound, location, temperature, etc.), which cannot meet such requirements.
- **Context processing:** Support applications with stronger context processing capabilities, e.g. context reasoning (knowing "car" is a subclass of "vehicle" helps an application interested in vehicle movement collect context information about cars) and context filtering (filtering certain context information for privacy purpose).

Existing studies on the middleware support mainly address the issues in the organization of and the collaboration amongst devices and services in the pervasive computing environment, but pay little attention to the design support of context-aware pervasive applications. None of proposed middleware infrastructures like *Gaia* [1], *EasyLiving* [2], *i-Land* [3], *Aura* [4] and *Interactive Workspaces* [5] can effectively assist application developers to handle all the above tasks.

Other studies focusing on context-awareness in [7][8][9] mainly analyze some useful features of context information and propose some helpful frameworks, yet still leaving the context processing duties to clients.

In this paper, we propose a novel approach based on ontology technology, and apply it in our *Cabot* project. Three important concepts, namely, context ontology, context pattern and context matching will be defined. Users use context patterns to subscribe their interested context information, while the middleware uses these context patterns to execute context matching for users. Context pattern helps implement flexible context gathering and classifying, and also contributes to enhancing applications with stronger context processing capabilities.

The remainder of this paper is organized as follows: Sec. 2 introduces related work in recent years; Sec. 3 presents the *Cabot* project – a software infrastructure supporting context-aware pervasive applications built on ontology technology; Sec. 4 further talks about some relevant issues about *Cabot*; Sec. 5 is a case study; and the last section concludes our contributions and explores future work.

2 Related Work

Existing studies on context-awareness are mostly concerned with either the frameworks that support the abstraction of context information or the context models that support data queries. Some typical works includes Cooltown project [7], Sentient Computing project [8] and Owl context service [9]. Their proposed context models generally lack formal bases; some of them even ignore the temporal aspects of context information.

Published research projects in the middleware support for pervasive computing include *Gaia*, *EasyLiving*, *i-Land*, *Aura* and *Interactive Workspaces*.

Gaia is a middleware project focusing on general-purpose pervasive environment. It makes use of active spaces [1] to encapsulate all low-level devices and services to provide a uniform interface such that developers can utilize and control the pervasive computing environment more easily. *Aura* is similar to *Gaia*, but uses a different approach. *Aura* has a context observer to monitor environmental changes that would trigger *Aura* to perform pre-defined actions. Each environment is managed by a distinct *Aura* system, and multiple *Aura* systems can cooperate to perform tasks.

i-Land works in a special environment that consists of a *DynaWall*, an *InteracTable* and a *CommChair* [3]. *DynaWall* is a wall-size touch screen, while *InteracTable* is a display on table. *CommChair* is a chair with computer network support. All devices can interact with each other and serve for presentations and discussions. *Interactive Workspaces* is another project sharing the same objectives with *i-Land*. It mainly focuses on the collaboration between a PDA and large screen projectors.

EasyLiving is a computer-centric system focusing on the living environment. A typical living environment has projectors, wireless keyboards, mice, finger-print recognizers, cameras, etc. Cameras can capture events in the house, and the images will be used for recognizing people and tracing their locations.

These projects work on the management of computing resources, while *Cabot* focuses on how to flexibly gather and classify context information and make further processing including context reasoning and context filtering.

3 Cabot System Architecture

In *Cabot*'s point of view, a complete pervasive computing environment is composed of Application Layer, Middleware Layer and Sensor Layer (Fig. 1).

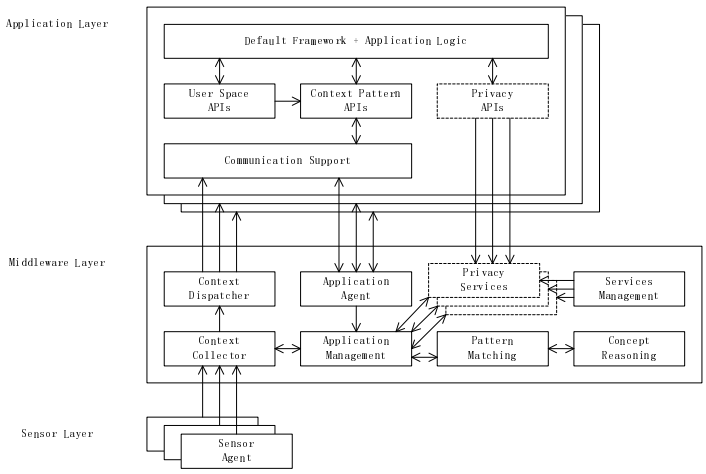

Fig. 1. The *Cabot* system architecture

Context-aware pervasive applications run at the Application Layer. This layer has complete client support in terms of *API*s. Applications can use context pattern *API*s to

manage (subscribe, update or remove) their own context patterns. Other *API*s include user space *API*s and privacy *API*s. They are related to user space management and privacy services respectively. An application framework is provided for application development. Usually, users do not have to pay attention to the details of communication with the middleware. They only need to focus on application logics, that is, make clear what their interested context information is and how to handle it.

The Middleware Layer is the kernel part. This layer implements five fundamental functionalities: (a) **application management** to be in charge of all registered applications, (b) **context pattern management** to be responsible for context pattern manipulations, (c) **context pattern matching** to be invoked automatically when the middleware receives any incoming context information, (d) **context semantics reasoning** to infer the semantic relations between concepts for reasoning, and (e) **third-party service management** to allow the middleware to integrate external context filtering services (e.g. privacy services) such that further context processing can be facilitated. The privacy services currently provided allows to modify or to hide some certain kinds of context information based on user identities and relevant privacy policies.

A concept related to the Sensor Layer is active entity. Active entities can be physical devices, software components or human beings. They periodically or non-periodically send context information to the middleware. Physical devices collect sensed context information (e.g., Tom enters into office 4208); software components generate derived context information (e.g., Cindy is busy); and human beings supply profiled context information (e.g., Cedric is supervised by Prof. Cheung). We regard each "qualified" active entity as a sensor agent. By "qualified", we mean that each active entity can exchange context information with the middleware based on a pre-defined context ontology.

4 Main Cabot Features

4.1 Context Ontology and Context Pattern

Most middleware infrastructures have limitations in supporting applications to flexibly subscribe context information. Usually, context subscription is based on context type. It may be inconvenient when users want to gather the context information mentioned in Sec. 1. Due to lack of the necessary support, users have to gather all relevant context information, and do analysis by themselves. This increases the network traffic in context transmission and the analysis workload in context processing.

Our approach is based on ontology technology. We propose context ontology, an ontology document catered for the pervasive computing environment. The context ontology acts as the context information agreement to which all applications, sensor agents and the middleware should conform in pervasive computing. Fig. 2 illustrates some major concepts (classes) and relations (properties) in the context ontology.

An **environment context** is defined by instantiating each ontology concept. When only part of ontology concepts is instantiated, it is called a **context pattern**. Applications subscribe their interested environment contexts to the middleware by means of context patterns.

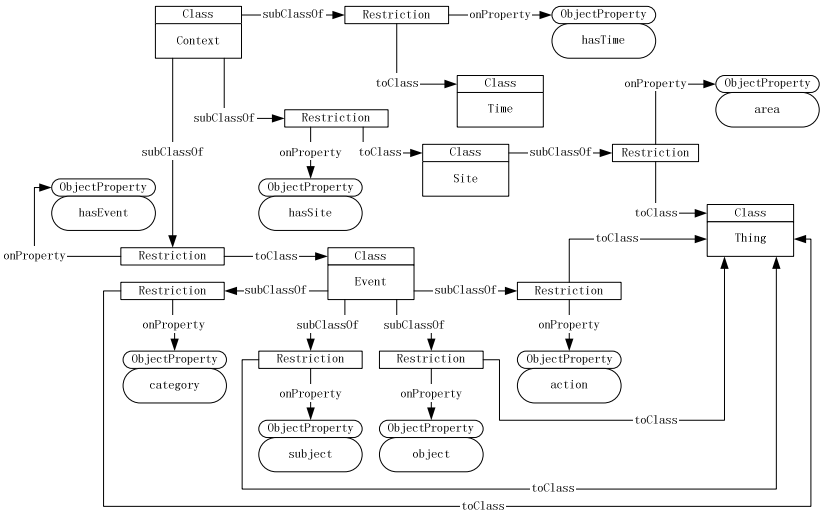

Fig. 2. The context ontology

4.2 Context Matching and Concept Semantics Reasoning

Cabot performs context matching between received environment contexts and subscribed context patterns. Both of them are transmitted, stored and processed in *XML* documents in practice. So an efficient tool for managing *XML* documents and an expressive language for describing matching rules are imminent. We utilize *xlinkit* to perform context matching. It is a software framework for checking the consistency of distributed *XML* documents. It comprises a rule language based on First Order Logic (*FOL*) and *XPath* notation [6]. For each incoming environment context, *xlinkit* checks whether it can be matched for any context pattern stored in the pattern repository according to predefined matching rules. The matching rules are written like this:

```
<forall var="context" in="/Context">
  <not><exists var="pattern" in="/Repository/hasPattern/Pattern">
    .....
  </exists></not>
</forall>
```

The omitted part is the kernel matching criteria that can be classified into three modes: exact matching mode, equivalent matching mode and plug-in matching mode.

If we require that a matching is recognized when a concept has exactly the same value in both the environment context and the context pattern, it is called **exact matching mode**. In the **equivalent matching mode**, the semantics relation between two concepts is identified to check equivalence. For example, when “weather” and “climate” or “enter” and “come into" appear in pairs, a matching is recognized. The **plug-in matching mode** further allows a context pattern to concern richer context information. When a more specific concept (say “car”) encounters a more general concept (say “vehicle”), this mode accepts it. The context matching example in Fig. 3 adopts all the three matching modes.

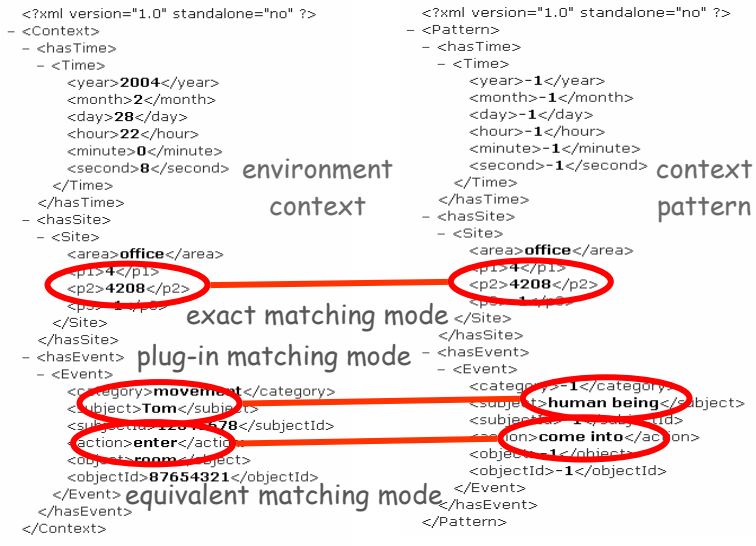

Fig. 3. A context matching example

When all concepts between an environment context and a context pattern are matched, *Cabot* asserts this environment context to be "qualified" for this context pattern.

The use of *xlinkit*'s built-in comparison operators is not enough for supporting context matching. So an operator special for concept semantics reasoning is required in *Cabot* implementation. This operator acts as the interface of a concept semantics reasoning subsystem built on a pervasive computing domain ontology database.

The domain ontology database stores much knowledge on semantics relations between concepts used in the pervasive computing environment. For example, "weather" is similar to "climate", and "car" is a subclass of "vehicle". Based on the domain ontology, the reasoning subsystem infers the semantics relation between two concepts as **equivalent**, **subsumed**, **including**, **intersecting** or **disjoint**.

The inferred semantics relation is the foundation of context matching. Let a concept in the environment context be c_1, and the counterpart in the context pattern be c_2:

- **Exact matching:** c_1 and c_2 are said to be matched when they are exactly the same;
- **Equivalent matching:** c_1 and c_2 are said to be matched when they are exactly the same, or have an equivalent relation;
- **Plug-in matching:** c_1 and c_2 are said to be matched when they are exactly the same, or have an equivalent or subsumed relation.

Some knowledge on concept semantics relations (e.g., "desk" is similar to "table") helps implement some special tasks (e.g., monitoring the abnormal movement of table-like things). Another usage of the ontology reasoning is to alleviate the naming problem across different sensor agents. For example, having known that "light" is similar to "lighting/ray/beam", a light-detecting application can behave better when facing different naming standards. In order to have applications enhanced with some certain reasoning capability, *Cabot* needs to incorporate the corresponding ontology related to the targeted application scenario.

4.3 User Space Switching and Application Framework

Available resources in pervasive computing are inclined to change. This could affect applications unexpectedly. *Cabot* allows switching of user spaces to help applications adapt themselves to the changeable environment. Each **user space** represents a space that contains context information relevant to the context patterns of this user space.

Cabot also provides a default application framework. This framework utilizes the *Cabot API*s to set up an asynchronous and context-driven programming model that adopts context subscription and callback handling technology.

5 A Case Study

Fig. 4 illustrates a computing environment. Room A is a printing room, Room B is a computer barn, and Room C is another computer barn. Any user to Room B or Room C will pass the Gate first.

An administrator, Peter, responsible for equipment maintenance usually stays in Room A, supplying printer paper when necessary and monitoring the coming users. Sometime, he goes to Room A and Room B to check whether everything is going well.

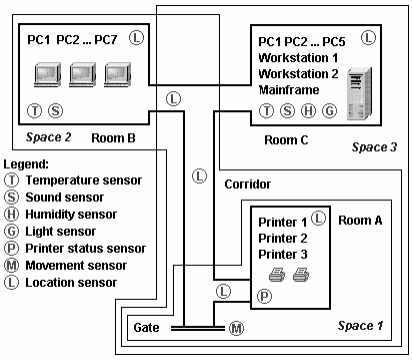

Fig. 4. A practical case

Suppose that temperature and sound context information is required to evaluate the PC status in Room B. But for Room C, additional humidity and light context information is also needed. Peter hopes to know the current equipment status once entering Room B or Room C, and no matter in which room he is resident, continuous monitoring of printers and coming users is expected.

We assume that all required sensors have been installed properly (Fig. 4). The following is the application design solution that comprises three user spaces (Fig. 4):

- **Space 1:** (Gate + Room A) Activating condition: Peter leaves Room B or Room C. Context patterns: (1) printer (area: Room A, subject: printer), and (2) people (area: Gate, category: movement, subject: people, action: enter).
- **Space 2:** (Gate + Room A + Room B) Activating condition: Peter enters Room B. Extra context patterns (to Space 1): (1) temperature (area: Room B, category: temperature, subject: PC), and (2) sound (area: Room B, category: sound, subject: PC).
- **Space 3:** (Gate + Room A + Room C) Activating condition: Peter enters Room C. Extra context patterns (to Space 1): (1) temperature (area: Room C, category: temperature, subject: computer), (2) sound (area: Room C, category: sound, subject: computer), (3) humidity (area: Room C, category: humidity, subject: air), and (4) brightness (area: Room C, category: light, subject: fluorescent lamp).

Cabot supports this application with two distinct capabilities: (1) context reasoning (e.g. “someone comes into …” = “somebody enters …”); (2) context subscription with plug-in matching (e.g. “computer” = “PC” + “workstation” + “mainframe” in Space 3).

6 Conclusions and Future Work

In this paper, we have overviewed several existing middleware infrastructures for pervasive computing. Their supports of context management are inadequate. To address this problem, we develop *Cabot* with the use of ontology technology.

A useful concept, context pattern, is introduced into *Cabot* to facilitate the context gathering, classifying and processing. In order to alleviate the naming problem and to enhance the expressiveness of context patterns, *Cabot* supports three flexible context matching modes. *Cabot* also allows the automatic and manual switching between user spaces to help realize adaptable context-aware pervasive applications.

At present, Cabot is still at a prototype stage. New functionalities and features (e.g. context trigger and context deriving) will be incorporated into the future releases of *Cabot*.

Reference

[1] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, K. Nahrstedt, Gaia: A Middleware Infrastructure to Enable Active Spaces, in *IEEE Pervasive Computing*, pp. 74-83, Oct-Dec 2002.

[2] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, EasyLiving: Technologies for Intelligent Environments, presented at Handheld and Ubiquitous Computing (HUC), Bristol, England, 2000.

[3] P. Tandler, Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices, at Ubicomp 2001: Ubiquitous Computing, Atlanta, Georgia, 2001.

[4] J. P. Sousa and D. Garlan, Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments, presented at IEEE Conference on Software Architecture, Montreal, 2002.

[5] A. Fox, B. Johanson, P. Hanrahan, and T. Winograd, Integrating Information Appliances into an Interactive Workspace, *IEEE Computer Graphics & Applications*, vol. 20, 2000.

[6] C. Nentwich, W. Emmerich, A. Finkelstein, Consistency Management with Repair Actions, in the *Proc. of the 25th International Conference on Software Engineering (ICSE'03)*, Portland, Oregon, USA, May 2003.

[7] T. Kindberg, et al, People, Places, Things: Web Presence for the Real World, *Technical Report HPL-2000-16*, Hewlett-Packard Labs, 2000.

[8] A. Harter, et al, The Anatomy of a Context-Aware Application, in *Mobile Computing and Networking*, pages 59-68, 1999.

[9] M. Ebling, G.D.H. Hunt, H. Lei, Issues for Context Services for Pervasive Computing, in *Middleware 2001 Workshop on Middleware for Mobile Computing*, Heidelberg, 2001.