A Uniform Device Information Access for Context-aware Middleware

Weiping Li, Weijie Chu, Frank Tung, Zhonghai Wu
School of Software and Microelectronics, Peking University,
Beijing, China, 100871
(wpli, chuwj, fctung, wuzh)@ss.pku.edu.cn

Abstract

This paper presents a middleware for building context-aware applications. One of the main components, Device Information Access (DIA), is discussed in detail. Since many kinds of devices (e.g., RFID, GPS, Bluetooth, etc.) can be used to collect the context information, the middleware defines the Device Information Access component to communicate with different devices. A set of interfaces are devised in DIA, and the common functions such as getting and setting a data element are defined in the interfaces. For each device, we shall provide an implementation of the interfaces to communicate with the corresponding servers or software agents. DIA can communicate with the software agents or servers using various protocols such as RMI, Web Services, and REST. In this way the access to the hardware are encapsulated by the middleware and virtualized to the end-point applications. The architecture of the middleware and the functions of DIA are discussed, and an empirical application is also developed to validate our design.

Key words: Context-aware services, Middleware, Device Information Access.

1. Introduction

Advances in the area of *internet of things* have made context-aware computing an emerging application paradigm. Context-aware applications could adapt their behavior automatically to the environment changes (e.g., location, temperature) by modeling explicitly these environment factors as context attributes [1]. An important next step for context aware computing is to develop context-aware middleware to facilitate context-aware applications. A context-aware middleware usually collects context information from sensing devices and applications, manipulates them and delivers the required contexts to the corresponding applications. With the aid of context-aware middleware, applications only need to deal with the context attributes provided by the middleware and adapt their behaviors accordingly [2].

Currently, many context-aware middleware has been proposed to aid the development of context-aware applications. Context Toolkit [3] is a well-known infrastructure supporting the development and execution of context-aware applications. By introducing context widgets for collecting, synthesizing and interpreting contexts, the middleware decouples the collection and use of contexts to facilitate the development of context-aware applications. Lime [4] is another one that supports both logical and physical mobility of agents in context-aware environment. It provides a coordination model for agents to share their contexts and collaborate seamlessly through the concept of tuple space. Cabot [5] provides additional context inconsistency detection and resolution services besides supporting context-awareness. In addition, there are also many similar context-aware efforts on middleware for pervasive computing [6] [7]. K. Kjær provides a survey of context-aware middleware and taxonomy to categorize them [8].

In context-aware middleware, many sensing technologies, such as RFID, Bluetooth, and GPS, can be used to get the context information. In our ongoing research project a context-aware middleware is defined. The Device Information Access module, as proposed, is the components which interacts with the devices and exchanges the data between the middleware and the devices. DIA communicate with the software agents of the devices, or simply a software component representing a certain information source, say, a weather forecast service from Internet.

The rest of this paper is organized as follows: Section 2 describes the proposed context-aware middleware. Section 3 presents the details of the device information access. Section 4 gives an example to illustrate the middleware as well as the DIA. Finally, section 5 concludes this paper and outlines future work.

2 The context-aware middleware

This section gives an overview of the context-aware middleware. The 'user' of the middleware is not the end user, but the software component that may communicate with the middleware. The architecture and the functions are also given.

2.1 The users

This section describes the users for the middleware in terms of its functions. The user here is neither the end user of a context-aware application nor the programmer who will use the APIs of the middleware to develop a context-aware application, but the one that uses the middleware. Hence typically the user of the middleware is the context-aware application system, the external device (e.g. Bluetooth) manager or the system administrator.

1. The context-aware application system relies on the middleware to provide the functions such as context management, send out some data objects to persons or hardware devices.
2. The device manager is the agent of the hardware devices or the software components that communicate with the middleware. It may send data or event to the middleware.
3. The system administrator is the person who is responsible for the configuration and maintenance of the middleware. He or she would build the context information database and the rules according to the context model.

2.2 The architecture of the context-aware middleware

The architecture of the context-aware middleware is shown in figure 1. With the middleware, one can develop his own applications. There are two repositories that work together with the middleware, namely, service repository and application specific context repository. The context model of a given application is manifested in the application repository shown on the figure 1, but the middleware itself is independent of the application context model. For example, the context model could be ontology based model, rule based model or ad hoc model. The main functions of every part are shown as follows:

1) Application Specific Context Repository

The context repository stores the real time context information as well as the rules used in a specific application.

- Context information database keeps the context information according to a given context model.
- Rule repository stores the rules that will be used in the context reasoning.

The context-aware application developers should define both the context model and the reasoning rules. In the runtime, Context Information Collection will get the real time context information and store it into the context information database. Surely the context reasoning may also get some deduced context information and deduced rules.

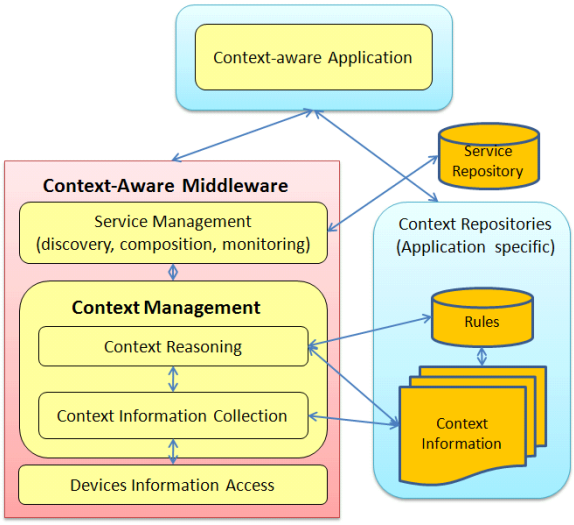

Figure 1 the architecture of the context-aware middleware–developer's perspective

2) Service Repository

Service Repository stores the services used in the application system. The basic functions of the service repository include service registry, service maintenance, service discovery, and service composition. The services may be developed by the application developers or come from some third party service providers.

3) Device Information Access (DIA)

DIA is the interface between the context-aware middleware and the external devices. The device here refers to not only the hardware device but also any software component that can provide context information. The functions of DIA include:

- GET information from devices
- PUT information to devices
- SET the configuration information to the devices.
- Event handler: With the basic functions of DIA, i.e. GET, PUT and SET functions, event handler provides two ways of getting specific information from the devices: polling and event driven trigger.

4) Context Information Collection (CIC)

CIC gets the real time context information from DIA and puts it into the context information repository. CIC also provides interfaces with other components, such as Context reasoning, Service Management and even the application.

5) Context Reasoning (CR)

CR will perform some inferencing on the context information and the rule information when any new context information is added or when the application manager triggers it. CR will try to guess what the current situation is and the result can be further used by the application system to provide proper service(s) to the end user. CR can add some deduced context and rules into the application repository. In this way the application repository will get more and more complete and accurate.

6) Service Management

Service Management will dynamically provide the services according to the current situation. The main functions are service registry, service discovery, service composition, and service monitoring. There are two ways to provide services–active and passive. When any event happens and the context changes, the service will actively provide services to the end user according to the reasoning result. The other is the passive way, in which the application or the end users request some services from the middleware.

3 The devices information access

DIA is the interface between the middleware and the external devices. The main function of DIA is creating a technology-independent, high-level application programming interface for two-way communication between external devices and the middleware. Based on DIA, technology-dependent attributes of the underlying technologies can be masked. Context information consists of many factors, including physical location, weather conditions, time of day, date and person's health status, etc. These are some of the example of important context information in context-aware systems.

3.1The functions of DIA

In DIA, we try to standardize a set of high-level interfaces which are built upon context information from heterogeneous context sources. So far we have implemented a variety of APIs ranging from location, weather, and health. To demonstrate this kind of API, we shall present a more detail description of the location API.

DIA Location API

The DIA Location API provides the following two main functions:

1) Supporting Request-Response pattern location information of the devices. The device's location is computed in response to an upper layer query.
2) Supporting event listener and subscription pattern for location information. The upper layer can request to be notified whenever a certain location-based event occurs.

This section describes the object model of DIA Location API shown in Figure 2.

Location: the Location class represents the location information of device; the location is represented by the Location object that contains an optional Coordinates object representing the geographical coordinates (latitude, longitude and altitude) and information about their accuracy, a timestamp and possibly information about speed and course of the terminal. The Location gives the accuracy of the coordinates as the radius of a circular area indicating the confidence level.

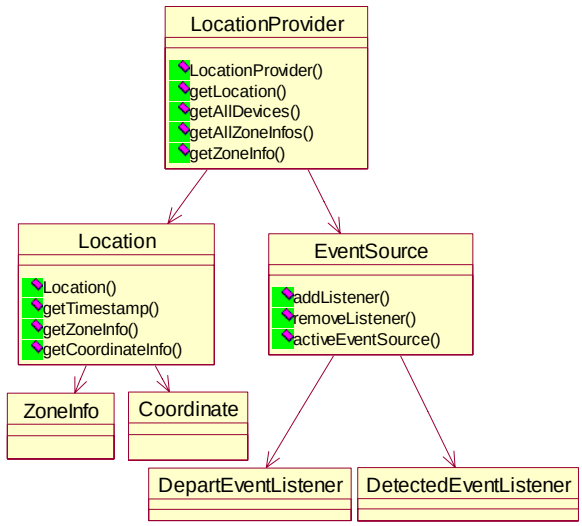

Figure 2 the object model of the DIA Location API

ZoneInfo: the ZoneInfo class represents an area. We partition real environments into several zones, each assigned to a given area of interest. Each zone is assigned a set of attributes, including a name, a set of optional coordinates, a geometric shape description, and a textual description. ZoneInfo are stored into a persistent repository, called ZoneInfoStore. The ZoneInfoStore class provides methods to create, update, retrieve and delete ZoneInfo from a persistent ZoneInfo store.

Coordinate: The Coordinate class represents geometric coordinates as latitude-longitude-altitude values. The values are expressed in degrees using floating values.

LocationProvider: The LocationProvider class represents a module that is able to determine the location of the terminal. This may be implemented by using existing location methods, including satellite based methods like GPS, and short-range positioning methods like Bluetooth Local Positioning. The application can select the location provider and obtain a LocationProvider instance that is able to fulfill requirements as closely as possible. By using the LocationProvider, the application can get Location objects representing the location of the terminal at the time of the measurement.

EventSource: The EventSource class represents the event source of all the events occurs in our middleware. We have defined two events in the events source, one is DeviceDepartEvent, and the other is DeviceDetectedEvent. The DeviceDepartEvent class represents the "One terminal departs from the zone" event. The DeviceDetectedEvent class represents the "One terminal is detected by the zone" event. The EventSource class includes three operations: AddListener, RemoveListener and ActiveEventSource. AddListener operation means add EventListener to the event source. RemoveListener operation means remove the EventListener from the event source. ActivateEventSource operation makes the event source activated and informs the event listener when correlative event occurs.

The implementation of the DIA interfaces is deeply depended on the technologies and devices. Usually for each kind of device there is a software agent for it. For instance, a Bluetooth server will manage some Bluetooth devices, an RFID middleware will communicate with many RFID readers, and so on. Mostly, the software agents are distributed with the context-aware application system. For instance, in the airport location-based system in section 4, the Bluetooth servers are deployed in different airports. In that case, the DIA is implemented with the Java Remote Method Invocation (RMI) to communicate with the Bluetooth servers. Certainly other technologies such as Web Services and REST can help to implement the DIA interfaces.

3.2 Data format in the middleware

This section describes uniform data format in the DIA location component. The different positioning implementations have heterogeneous data format for location information. The DIA components hide this heterogeneity of the various positioning technologies from the upper layers. One important task is the mapping between different locations data formats if the request specifies a different format than provided by the selected technology. In this case, DIA components use Hybrid location models [9] (combining symbolic and geometric coordinates models) to represent location information.

In DIA Location API, the ZoneInfo class represents hybrid location information. A given ZoneInfo is an interested area. Each zone is assigned a set of attributes, including a name, a set of optional coordinates, a geometric shape description, and a textual description.

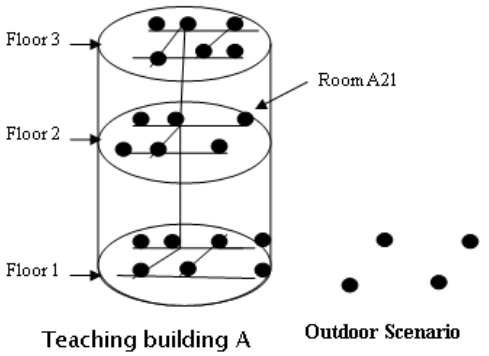

Figure 3 a location model

Figure 3 shows an example of location, teaching building A is located in our school, and room A21 is our office. The location information of room A21 is described in table 1.

Table 1 The Location info of roomA21

	Name :Rome A21
Coordinates	Latitude : 61.448
	Longitude : 23.885
	Altitude: Null
ZoneInfo	Street: No.24 Jinyuan
	City: Beijing
	Country: China
Description	BuildingName: Teaching Building
	RoomName: A21
	Floor:2
	NeighbourRoom: A22
	NeighbourRoom: A20

4 An example application

To evaluate the functionality and performance of the middleware, this section presents an example to

show how to establish a context-aware application based on our middleware. In this airport example the Bluetooth technology is used for getting the location information. A simple context model is built to analysis the behavior of the passengers and recommends proper services to the passengers.

4.1 Smart airport scenario description

The experiments reported in this paper are based on the following scenario. In an airport, Mr. Mo is a passenger to fly from Beijing to Shanghai at 12:00. He arrives at the airport at 11:05. On his way to the airport, because he doesn't appear in the region of check-in one hour before his departure time; he receives a short message to remind him that he needs to check-in in a hurry. Mr. Mo checks in at 11:10 and then goes to the security check. His boarding time is 11:40. On his way to the gate, he meets a friend who has not seen for many years. They talk for a long time with each other and Mr. Mo forgets the boarding time. At 11:30, because he doesn't appear in the gate area ten minutes before boarding time, he receives a short message to remind him that he needs to go to the gate immediately. Thanks to this smart airport service, Mr. Mo boards his plane just in time.

The main context variables in this application are shown in table 2.

Table 2 Main Context variables

Context	Type	Comments
Time	time	Current time
Check-in location	Boolean	Whether the passenger appears in the check-in area
Boarding location	Boolean	Whether the passenger appears in the boarding area

4.2 Hardware deployment

To develop this system, we choose the BLIP [10], a Bluetooth-based positioning system, as context sensor to locate passengers. It is deployed in the area of check-in and boarding gate to detect whether the user appears in those regions. The area covered by a blip node (Bluetooth sensor inside) is called a zone and an airport will be divided into many such zones. Each blip node uses Received Signal Strength Indicator (RSSI) to detect the position of a passenger in its zone. The Blip nodes accuracy of up to $\pm$ 15 meters will therefore suffice.

4.3 Application's architecture

The system architecture of Smart Airport is shown in Figure 4.

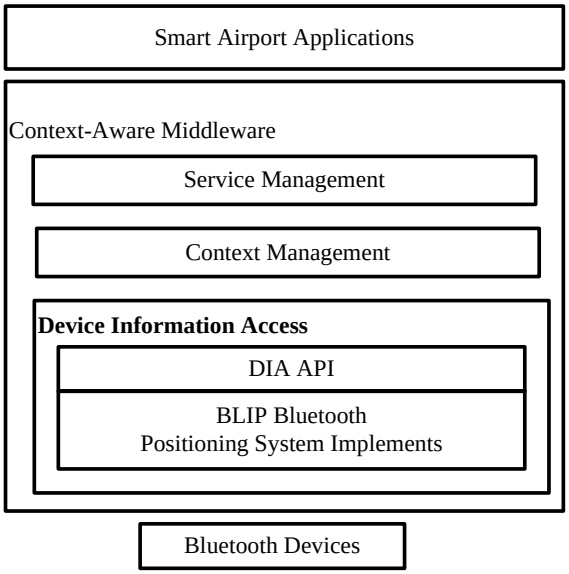

Figure 4. Smart Airport Application Architecture

4.4 Performance analysis

During the experiments, we found that Blip Bluetooth positioning system could detect almost all mobile phones that have Bluetooth activated within the zone. Also it can efficiently push the message to mobile phones with a short response time and a high performance rate.

Table 3.Testing Data of this experiment:

Phone's Type	Time phone entering the zone	Time finding phone	Time Difference
OPPO	21:50:13	21:50:17	4seconds
OPPO	21:50:50	21:50:59	9second
OPPO	21:52:22	21:52:28	6seconds
NOKIA	21:53:21	21:53:27	6 seconds
NOKIA	21:55:55	21:56:05	10 seconds
NOKIA	21:59:36	21:59:39	3 second

The application accesses the middleware API through the encapsulated Blip positioning system instead of calling Blip APIs directly. The result shows that DIA encapsulation did not cause delay of the system response time. The average time for detecting the phone through DIA and BLIP API is almost the same, 6.3 seconds. We show the experiment data in table 3.

5 Conclusions

This paper proposes a context-aware middleware with a more detailed discussion of one of the main components, DIA. In DIA we define a set of interfaces which exchanges data between the middleware and different devices. DIA leverages different protocols to communicate with remote devices. An experimental application is developed to validate the middleware and the result shows that this middleware provides the core functions for developing context-aware applications.

Next, more modules will be added into DIA to deal with new devices. The context model will be another focus, and we are currently working on an ontology-based context model and reasoning method to better collect and understand the context.

6 Acknowledgments

Research in this paper is supported by the Danish Strategic Research Council (Grant NO. 2106-08-0046) and the National High-Technology Research and Development Plan of China (863) under Grant NO. 2009AA04Z120.

7 References

[1] Yau, S.S., Wang, Y., and Karim, F. Development of Situation-Aware Application Software for Ubiquitous Computing Environments, In Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC 2002), pp. 233-238.

[2] Chunyang Ye，S.C. Cheung，et al, A Study on the Replaceability of Context-aware Middleware, Proceedings of the First Asia-Pacific Symposium on Internetware, 2009, Beijing, China

[3] Dey, A.K., Abowd, G.D., and Salber, D. 1999. A Context-Based Infrastructure for Smart Environments. In Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments, Dublin, Ireland, December 1999, pp.114-128.

[4] Murphy, A.L., Picco, G.P., and Roman, G-C. LIME: a coordination model and middleware supporting mobility of hosts and agents. ACM Transactions on Software Engineering and Methodology, vol.15, no.3, July 2006, pp. 279-328.

[5] Xu, C., and Cheung, S.C. Inconsistency Detection and Resolution for Context-Aware Middleware Support. In ACM SIGSOFT Symposium on the Foundations of Software Engineering, Sep 2005, pp. 336-345

[6] Bellavista, P., Corradi, A., Montanari, R., and Stefanelli, C., Context-aware middleware for resource management in the wireless Internet. IEEE Transactions on Software Engineering, vol.29, no.12, Dec. 2003, pp. 1086-1099

[7] Capra, L., Emmerich, W., and Mascolo, C. CARISMA: Context-Aware Reflective Middleware System for Mobile Applications. IEEE Transactions on Software Engineering, vol. 29, no.10, Oct 2003, pp. 929-945.

[8] Kristian Ellebæk Kjær, A SURVEY OF CONTEXT-AWARE MIDDLEWARE, in Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering, 2007, Innsbruck, Austria, pp 148-155.

[9] Becker C, Durr F, On location models for ubiquitous computing. Personal and Ubiquitous Computing (Lecture Notes in Computer Science, vol. 9, No.1). Springer London, 2005; 20–31.

[10] http://www.blipsystems.com/