WaterCOM: An Architecture Model of Context-oriented Middleware

Keling Da, Marc Dalmau, Philippe Roose

WaterCOM: An Architecture Model of Context-Oriented Middleware

Keling DA
IUT Bayonne
LIUPPA, UPPA
Bayonne, France
Email: kda@univ-pau.fr

Marc DALMAU
IUT Bayonne
LIUPPA, UPPA
Bayonne, France
Email: dalmau@univ-pau.fr

Philippe ROOSE
IUT Bayonne
LIUPPA, UPPA
Bayonne, France
Email: roose@univ-pau.fr

***Abstract*—Integrating physical and information space into applications increases application's complexity and development difficulty. In Ubiquitous environment, context collection, aggregation and notification raise complex scientific problems and new challenges. In this paper we address these challenges by proposing a conceptual context-oriented middleware architecture. We first discuss the reason to use context in ubiquitous computing, and context-oriented middleware requirements. Then we present our approach by describing a service-oriented architecture model. It provides a dynamic adaptation ability, supports multiple context models and multi-domain context consumer. Finally, we discuss the benefit of our conceptual approach by describing and comparing current context middlewares.**

***Keywords*-Context-awareness; Middleware; Ubiquitous computing; Software architecture;**

I. Introduction

Ubiquitous computing was first proposed by Mark Weiser in September 1991 as: "Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user" [1].

According to the previous definition, ubiquitous computing must be pervasiveness, convenience and adaptable. The future ubiquitous applications face with heterogeneous and dynamic environments. They must be able to adapt (supervision/self-*) and react dynamically to the environment (heterogeneous hardware and software environment) and to the context [2] (user and environment context).

Context middleware is one solution to collect low-level context information (e.g. GPS position, temperature of environment, time and date, etc) and provide high-level context information (e.g. someone come into the meeting room, the meeting is interrupted; Mike drives his car to go to work; etc) more suitable to make long term and pertinent decisions. In ubiquitous environments, the context middleware needs to take care of the context collection, and aggregation to provide a high-level context model reasoning, and a context notification mechanism.

The paper structure is as follows: Section 2 discusses motivation and objectives via the definition of the context, the motivation for context-oriented middleware and its definition. Section 3 presents details of our conceptual architecture approach. Section 4 describes and compares current context middlewares and discusses the benefits of our approach. Section 5 concludes the paper.

II. Motivation

A. What is Context?

The context includes the operating environment and user's utilization environment. The operating environment includes any observable information (i.e. any environment information that can be captured and measured) by the software system, such as end-user input, external hardware devices sensors, program instrumentation, network infrastructure, etc [3]. The user environment means anything about the user, which includes user's profile intents, and user's context information.

B. Why get and use Context information is difficult?

To use context information, we firstly need to collect it. Then make it semantically interpretable, and therefore ready for analyze. To collect context information in a highly dynamic distributed environment is a challenge. According to the previous context definition, it is a very wide set of information. There are various types of context providers, and the environment is heterogenous. That raises the problem of '*how to collect context information from heterogenous sensors with minimum recurrence code and maximum code reuse*'. In addition, there is another problem in mobile environment, which is '*how to continue acquiring context information*'.

After collection, the use of raw data to serve analysis raises the following difficulties: i) It must be abstracted to make sense for context consumers. A GPS coordinate is meaningless for a map service application. It needs to be translated to road or building name, etc. ii) It must allow context consumer accessing to all information level. Context consumers have different emphases with context information. Some need raw data while others need interpreted data. iii) Context consumer needs a generic and uniform access interface. That will simplify context consumer's development and provide code reusability. To deal with such challenges and difficulties, context oriented middleware is one of the more powerful solutions.

The process of context information and the build of high-level context models consume computing resources, which may be critic on embedded and/or mobile devices (wireless sensors, smartphones, tablet, etc.). Changes in the environment must be detected at run time and therefore the high-level model must also be built in real time.

In pervasive environments, computing resources are limited. Context middleware needs to adapt itself to get aware of surrounding resources (e.g. benefit more powerful computing resources, save battery life, explicit information of neighbor, etc.) In the next section, we present our context-oriented middleware solution.

C. What is Context-oriented Middleware?

A Context-oriented Middleware (CM) is a middleware aware of its context. The context is both (contextual) information as well as the architecture of the application running above and of the CM itself. A CM provides mechanism to dynamically reconfigure the application and the CM itself. See our architecture model WaterCOM in figure 1.

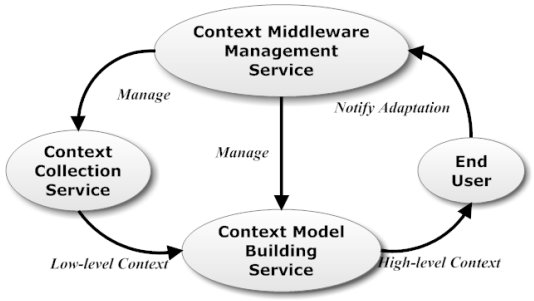


Figure 1: Architecture model WaterCOM

Our longterm research is to design and to implement an architecture of Adaptation System (AS). An adaptation system is a system providing application adaptation according to the environment. In an adaptation system, there is a middleware layer and an adaptation platform. An adaptation middleware layer composes adaptation middleware, context middleware and communication middleware. It provides some mechanisms to achieve adaptation tasks. It is also a layer that handles heterogeneity, communication and context collection.

In addition an adaptation platform is also called Decision Making system. A Decision Making system (DM system) reasons and identifies context situation that the CM provides according to high-level context model. If DM system decides to execute an adaptation operation, the CM could be part of the adaptation. It means that CM will be distributed into execution environment. To be able to accomplish this purpose, the architecture design of the CM should be distributed, with loose coupling and can be dynamically reconfigured (hot reconfiguration). Such CM architectures are called context-aware middleware.

Context Middleware has the following requirements: Firstly, sensors could be hardware sensors such as temperature sensor, or software sensors such as system information collection component (e.g. CPU performance monitor, network performance monitor, and memory usage monitor). Hardware sensors are called physical sensor and software sensors are logic sensors. CM provides a unified context information service to its end-users (i.e. context-aware application or Adaptation System) in a distributed environment. It hides the complexity and the heterogeneity of sensors. Secondly, in ubiquitous environment, context information data is continuously created. Different kinds of context users have different emphases. CM provides data management services and filter context notification/query services. It manages the data flow of contextual information. Finally, information from physical sensors, called low-level context is acquired. Without any further interpretation, it can be meaningless, trivial, vulnerable to small changes, or uncertain [4]. CM abstracts low level context information to provide high-level context models. These models are specific to context reasoning and situation identification.

III. Context-oriented middleware architecture

In this section, we introduce a generic conceptual Context-oriented Middleware Architecture Model for pervasive environment. It is based on a service-oriented architecture and aims these requirements mentioned in section 2.

A. Architecture Model

The model is based on three services: Context Collection Service (CCS), Context Model Building Service (CMBS) and Context middleware Management Service (CMS). (See figure 2). This architecture model allows our context-oriented middleware to support dynamic adaptation, multi-types of context model, and multi-domains of user. This architecture is based on Kalimucho middleware [2], which provides a runtime dynamic reconfiguration with distributed environment. We will use Kalimucho's Osagaia [5] component-based model to implement it.

Context Collection Service composes of various Context Collectors (CC (s)). Context Collectors are distributed on network and executed on various devices. A CC has two components, Sensor Monitor and Context Collection Communicator. Context Collection Service handles hardware and software heterogeneous in order to provide low-level context information. It is detailed in section 3.B.

Context Model Building Service composes of Aggregator, Interpreter Repository and Model Repository. Context Model Building Service builds instances of high-level context model. We introduce it in section 3.C.

Context middleware Management Servic composes of Low-level Context Manager, High-level Context Manager

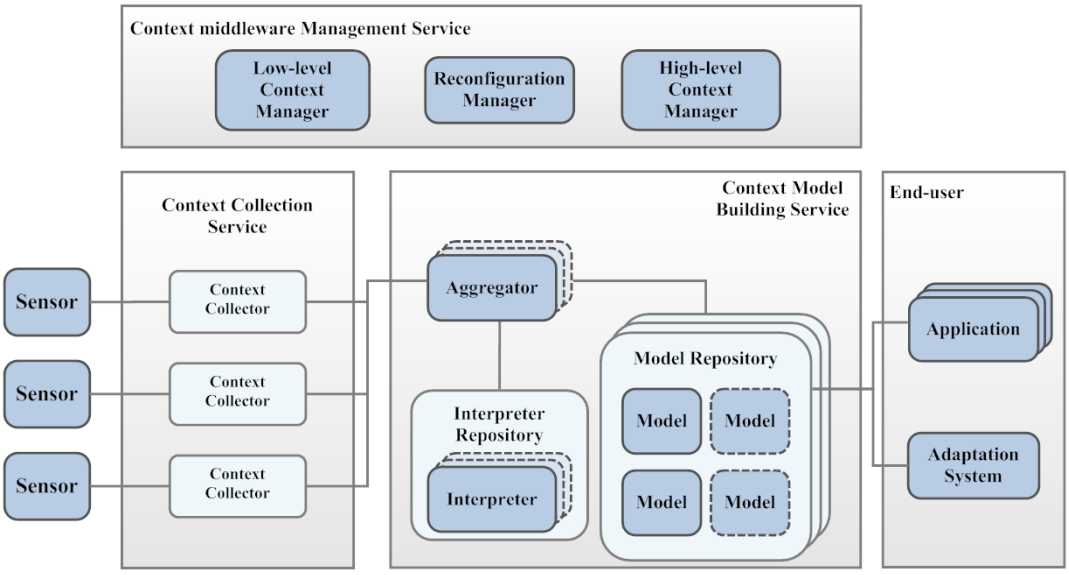


Figure 2: Architecture model.

and Reconfiguration Manager. Context middleware Management Servie handles context resource discovery and supervises component's status. We detail it in section 3.D.

We define a high-level context model building process for our architecture model. Context information is created by operating environment (e.g. Applications, OS, Hardware, etc.). Then, Context Collector observes context information and send to Aggregator to execute high-level model building process by using Interpreters. Model Repository keep an instance per model and when an instance is changed Model Repository will notify it to consumers (e.g. Applications, adaptation system, etc.). See figure 3.

First, to build a high-level context model, we need to reason on some low-level context data and interpret it. The question is, how to get this data from a distributed environment, and the condition to notify it (e.g. An application needs to be aware of a specific temperature information about room 10 at 9 o'clock each morning to construct its context model, but there are many temperature sensors online and they are in different location)? Low-level Context Manager provides two services to resolve the first problem, i) Context resource discovery service that allows Aggregator to find what context information it needs. ii) Context resource subscription service allows Context Collector to subscribe as a context resource. We give more detail about Low-level Context Manager in section 3.D.1. Context Collection Communicator allows context consumer to subscribe to a context notification. Context Collection Communicator notifies context data under user's given condition. (See section 3.B.2.)

Second, the end-user of CM needs to subscribe its Aggregator(s) in order to get its context model instance. High-level context manager works with Model Repository to provide a high-level context information notification/query service and Aggregator management. We introduce them in section 3.C and section 3.D.

Finally, how CM supports hot-reconfigurations? Our model is based on the Kalimucho middleware and its Osagaia component model as mentioned above making possible dynamic reconfiguration while the middleware is running. Hence the CM is natively reconfigurable during the runtime. However, during the runtime, each component needs to know its client's or cooperator's status (e.g. Context collector needs to know its subscribers status, if someone being migrated, Context collector will buffer the context data and send it after subscriber's migration). Reconfiguration Manager will handle each instance of component's status and information.

Next we will first discuss the Context Collection Service in detail.

B. Context Collection Service (CCS)

Context Collection Service is required in a distributed architecture because context may need to be acquired from different distributed sensors. CCS is a layer including many different and distributed Context Collectors (CC). One CC corresponds to one context source (e.g. a GPS sensor, a temperature sensor, a CPU monitor service, etc). CC is composed of two components: Sensor Monitor, and Context Collector Communicator. Sensor Monitor collects context information from sensors and controls sensors configuration and functional state. Context Collector Communicator is a unified communication service, hiding communication complexity to context consumers.

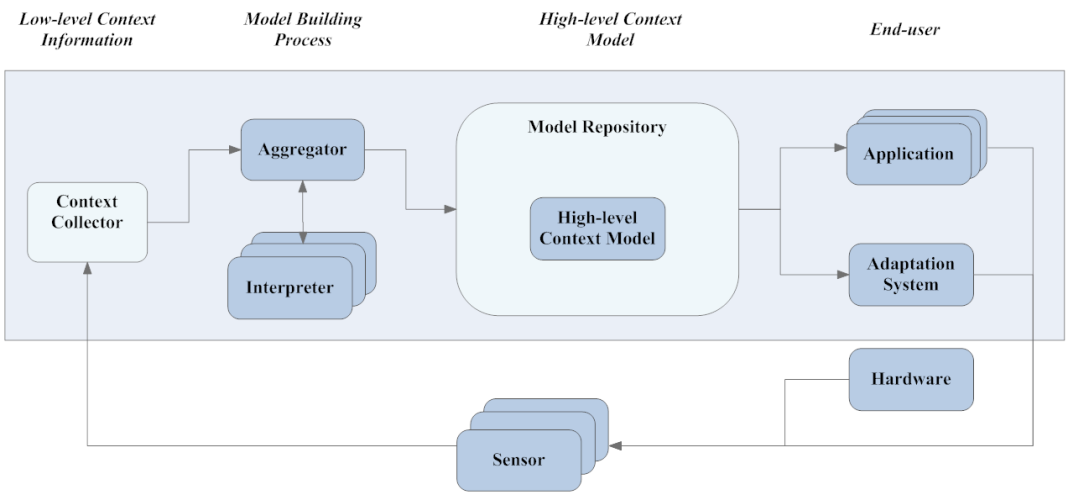


Figure 3: Context middleware high-level context model building process.

1) Sensor Monitor (SM): Sensor Monitor directly deals with (physical and logical) sensors API. It is like a data access layer. In addition, SM provides a sensor control interface to monitor and adjust sensors. The sensor API specifies implementation of a SM. However, SM provides a unique interface for all sensors. This interface includes: *connectSensor(), setSensorState() / getSensorState(), setSensorConfig() / getSensorConfig(),* and two exceptions: *ExSensorStateIncompatible and ExSensorConfigIncompatible*.

Sensor state is the functional state, i.e. *Start, Pause, and Stop*. Sensor configuration is a set of attributes depending on concrete sensor. A new configuration will adjust functional of sensor. For example: a temperature sensor provides temperature data in Kelvin unit. Context consumers may acquire other units like Celsius or Fahrenheit. Hence, Context consumer needs to configure CC using Context Collector Communicator. It will call *changeSensorConfig()* of SM.

2) Context Collector Communicator (CCC): Context consumer acquires context information from Context Collector by CCC. It means CCC is a unified communication interface between consumer and Context Collector. One CCC can deal with many context consumers. When context consumers subscribed to CCC, they receive notification when data is collected/updated. CCC deals with transparency of data transport. It also allows the transfer of Sensor configuration to Sensor Monitor. In addition, CCC provides a conditional notification. Consumer can set a notification condition like a notification frequency, a specific time, a specific location. For example, if temperature more than $30C^\circ$, collect temperature every 10 min; else once every hour.

To implement these functionalities, CCC needs to maintain a subscribers information table. It contains: consumer id, consumer communication address, consumer state, and notification condition as items. Notification condition is an option. Context consumer state can be indicated: available (i.e. waiting for notification), or busy (i.e. may be on reconfiguration process, CCC will keep notification data into buffers and send data while consumer state changed as available.), or unavailable (i.e. CCC will delete this subscriber from the table). Consumer can set its state, communication address, and notification condition by calling CCC's methods (e.g. *setConsumerState(), setConsumerAddr(), and setNotificationCondition()*).

CCS can also directly provide low-level context data to consumer. If consumer does not need the high-level context model reasoning, they can use context middleware as a context collection platform to reduce their development cost. Next section will introduce context model building service in detail.

C. Context Model Building Service (CMBS)

Context model building service get low-level context information from Context Collection Service and build instance of high-level context model than store it into Model Repository. CMBS has three components, Aggregator, Interpreter Repository and Model Repository.

1) Aggregator: Context-aware application needs to specify its own context interpretation process to build high-level context model. This process is defined at aggregator. Application or other context consumer can implement own aggregator and plugin it into the Context Middleware (i.e. Each aggregator needs to be subscribed at High-level context manager, we present it in section 3.D.2). Each context consumer (i.e. context-aware applications and others) has own interests of context. Hence, context consumer uses its own aggregator to subscribe to context information and to interprets it to build its context model.

The aggregator processes high-level context models. An aggregator may define the process like checking low-level data, interpreting data, building model, and saving model instance by using Model Repository. However, the building process depends on concrete context consumer's needs. In addition a context consumer can subscribe various aggregators and compose them to make the model instance. Implementation data verification can be based on various algorithms: Fuzzy Logic [6], Probabilistic logic [7]) or leaning based: Bayesian network [8], Hidden Markov Models [9], [10], and Dempster-Shafer theory[11]). It is called data checking interpreter , in our model. Interpreter could be any algorithms component that handles one context interpretation. Aggregator use these interpreters to reason on model, check data, and so on. Interpreters are stored in Interpreter Repository, which we will introduce next.

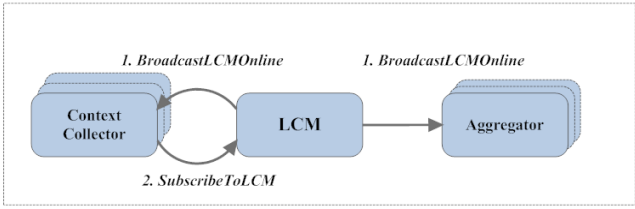
2) Interpreter Repository (IR): Interpreter Repository is a repository allowing aggregator to find and use interpreter to accomplish high-level model building process. IR provides a search interface to request interpreter and maintains execution of interpreters (i.e. Where and how to execute; which parameter).

Each Interpreter handles one objective of context interpretation. It interprets low-level context information to high-level context model. For example, a GPS sensor get longitude and latitude coordinate. We get a low-level context model by Context Collection Service, e.g. N $43\,°29.58072'$, W $1\,°28.49099'$, gpsid01, 2011-07-12. This data cannot be directly used. An application needs a more abstracted information like the name of the corresponding city. Its aggregator will use a GPS coordinate to Address Interpreter to interpret the context model. After interpretation, aggregator will get this: N $43\,°29.58072'$, W $1\,°28.49099'$, city: Bayonne, gpsid01, 2011-07-12. It is a simple example, an implementation of interpreter could be very complex. Interpreter can be implemented as a web-based interpreter or as a local interpreter. Interpreter can be reused and or aggregated in Context middleware by IR.

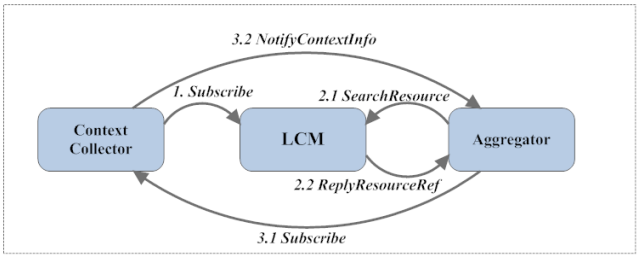
3) Model Repository (MR): Model Repository is responsible for maintaining a set of instance of context models provided by Aggregators. It provides a unified context model query/notification interfaces for context consumers. It is a context model specified component. (i.e. for different model, the MR implementation will be different.)Context middleware may have various MR instances. Thus, Context middleware may maintain different context model categories (e.g. ontology model, object model, spatial model, etc.). Each MR corresponds to one context model category.

D. Context middleware Management Service (CMS)

This service composes of three components: Low-level Context Manager, High-level Context Manager and Reconfiguration Manager.



(a) LCM Starts for the first time



(b) LCM Started

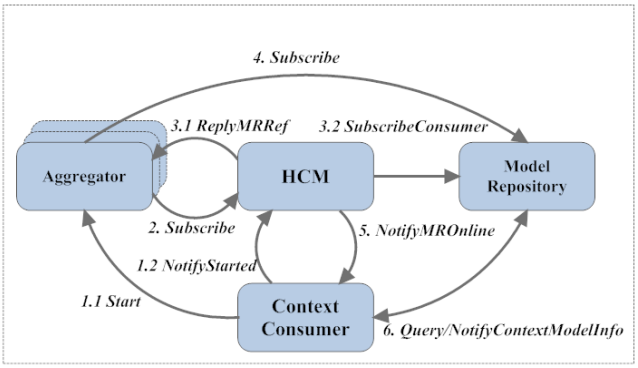
Figure 4: LCM works sequence

1) Low-level Context Manager (LCM): The low-level context manager provides Context resources subscription service and Context resources discovery service. Context resources subscription service allows Context Collector to subscribe to LCM. Context Collector subscribes with sensor id, device id, communication address, type of context data, and description of sensor. This information will be used to identify context resource. If a new Context Collector starts, it will subscribe to LCM. If it does not find LCM, means there is not LCM online, it will wait for a LCM's subscription notification and switch sensor to sleep state. When the LCM start, for the first time, it will broadcast a subscription notification for finding all available Context Collector on networks. (See figure 4a)

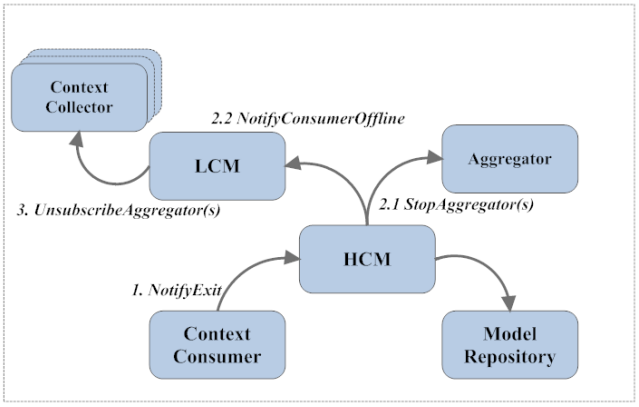
The LCM also provides a context resources discovery service to aggregator. Aggregators ask LCM to find one or more context resources by context data type, and resource description (e.g. it can be sensor id, device id, communication address, or anything matching with description of sensor or a combination of information). The LCM will firstly search into its context resources database. If it cannot find the demand context resource, it will broadcast a subscription notification. Finally, if there is no response, it will keep the demand as an unfinished task. The aggregator can cancel the demand, or wait for response, or change the resource description, or trigger an exception. It depends on context consumer's strategy.(See figure 4b)

2) High-level Context Manager (HCM): The HCM handles subscription of Aggregator and subscribes context consumer to MR. Applications or Adaptation System need to subscribe its aggregator(s) to context middleware by HCM. They can have one Aggregator or a group of Aggregator.

However they can only have just one instance of context model per each. When consumer subscribes its aggregator(s) to HCM, HCM also finds the matched MR and subscribes the MR to Aggregator(s). Then, HCM subscribes these aggregators to the MR. After this, the consumer can query its model, or when the instance is updated, the MR will notify it to the consumer. (See figure 5a)



(a) HCM Subscription Sequence



(b) HCM Cleaning Sequence

Figure 5: HCM works sequence

When context consumer is offline or stop, the HCM also handles the cleaning work. HCM will stop its aggregator(s), delete its model in model repository, and notify LCM to unsubscribe aggregator(s) from all context collector(s). (See figure 5b)

3) Reconfiguration Manager (RM): The Reconfiguration Manager works with Adaptation System. The Adaptation System sends a reconfiguration plan to RM before adaptation. This plan indicates which component will be migrated. RM will change their status as waiting for migrate'. When Adaptation System finish adaptation process, it will notify RM with all status and communication addresses of components. RM will update their information and notify component, which already corporate with it, to continue to work. (See figure 7)

E. Works with context consumer

The context consumer means applications and adaptation system. There is some differences to work with application and adaptation system. Context consumers use context middleware to build its context model instance. In addition, adaptation system adapts context middleware by migrating its components. Thus, how to build high-level context model with context middleware (see figure 6). Context consumers needs to implement their Aggregator(s) and or Interpreter(s) depend on what their needs. Then, subscribes them to context middleware.

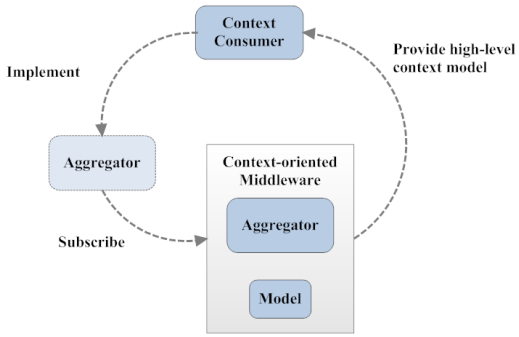


Figure 6: To build high-level context model.

To adapts context middleware, adaptation system needs to contact context middleware's Reconfiguration Manager to be sure that the adaptation will do it correctly. adaptation system prevents context middleware adaptation plan and sends adaptation results information to context middleware. (See figure 7)

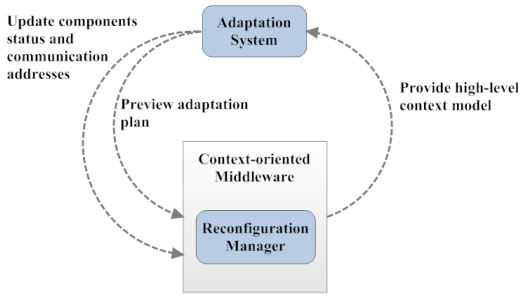


Figure 7: To adapts context middleware with adaptation system.

F. Summary

In this section, we introduced the high-level context model building process and three main services. Each main service is detailed with their components. At the end of this section, we also presented how an application works with our context middleware and how an adaptation system can work with it and adapt it. Next we will present some related works and compare them with our architecture model.

	Distributable	Support high-level model	Reconfigurable	Adaptable	Multi-model types	Multi-domain Interpretation
Context Toolkit	X	X	X			
SOCAM	X	X	X			X
SPACES	X	X				
WaterCOM	X	X	X	X	X	X

Table I: Related works comparing table

IV. Related works

In this section, we present works directly related to our proposal and point out the differences with our model.

Context Toolkit [12] is a framework that aims at developing reusable solutions to simplify the design and implementation of context-enabled applications. Context Toolkit adopts the widget concept from GUI (Graphic User Interface) toolkit that it is called Context Widget. Context Widgets are software components that is responsible for acquiring context information from sensors and they provide applications with access to context information from their operation environment. Context Widgets can be distributed across different machines. It can be composed to provide richer context information. Context Toolkit does not support Hot-reconfiguration and adaptation. However, we interest the idea of Widget. Our Context Collection Service has a similar design.

SOCAM [13] is a service-oriented middleware that uses ontology context model based on OWL. It has been designed to support the building of context-aware services. SOCAM proposes and uses CONON [14] ontology model that has two layers design that supports separation of concepts considering generality and specificity. SOCAM has a layered architecture that aims to provide an efficient infrastructure and it is a distributed middleware. It consists in different components such as: Context Providers abstract context sources, Context Interpreter provides logic reasoning service, Context Database stores context ontologies, and Service Locating service provides middleware components and context consumers to locate these services and their presence. We have a similar context processing process and similar services. Furthermore, our approach aims to provide a generic solution for context consumers and supporting consumers to use their own context model.

SPACES [15] is a lightweight middleware solution enabling the versatile and efficient mediation of context information. SPACES adopts COSMOS [16] framework as a scalable model for processing context information. COSMOS is a policy-based context processing framework. It processes context information by a composition of context nodes. Context node is a software component that responsible for acquire context information and interpretation of context. SPACES use REpresentational State Transfer (REST) to distribute COSMOS into ubiquitous environment. It is a part of CAPPUCINO [15] platform. We interest the composition of context nodes to provide the context processing. Aggregator can composites interpreters like their context nodes to processing context information. Our model allows composing aggregator to do more complex context processing.

Finally, all the above described middleware provide mechanism for dealing with inherent heterogeneity and complexity of ubiquitous environment. Hence, to compare to our proposal (see Table I), they do not: i) provide dynamic adaptability with ubiquitous environment (it can be adapted while working with adaptation systems), ii) provide multi-types model supporting, and iii) provide different domain specific context models interpretation. In ubiquitous environment, context consumers may have different emphases to select their context models, and that will lead to interpret these context models to use different interpretation algorithms. Our approach deals with accessing different context models in a transparent and uniform way, allowing context consumers to best use context resources, and simplifying application code and reusing interpretation algorithms.

V. Conclusion

This paper presents a conceptual generic architecture model for context-oriented middleware. It provides a context communication service for dynamic and mobile distributed environment. It allows context consumers to get/retrieve and act on its high-level context information locally or remotely. The high-level context information is provided by context model building service. Context consumer can search context resources by using context middleware management service and subscribe as low-level context information consumer to directly get low-level information by context collection service. Context collectors are distributed and support dynamic reconfiguration. It hides software and/or hardware complexity and heterogeneity. We already implemented low-level context manager, context collector and reconfiguration manager on Kalimucho[1] and we will continue to develop the missing items.

As future work, we plan further context model history service, which is called Context History Manager. Context History Manager will persistence context models into database (local/ remote/ cloud). It can restore any release of context model, i.e. a complete history context model restoration.

It provides a blackboard mechanism to store context information, which will be used to context reasoning such

[1]Kalimucho OW2: http://www.ow2.org/view/ActivitiesDashboard/Kalimucho

as probabilistic reasoning or leaning based context reasoning. This kind of information directly comes from context collection service. It provides a context history management interface to high-level application to manage their context model.

REFERENCES

[1] M. Weiser, "Some computer science issues in ubiquitous computing," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, Jul. 1999. [Online]. Available: http://dx.doi.org/10.1145/329124.329127

[2] C. Cassagnes, P. Roose, M. Dalmau, and C. Louberry, "Kalimucho : software architecture for limited mobile devices," *ACM SIGBED Review, Special Issue on the - 2nd Workshop on Adaptive and Reconfigurable Embedded System*, 2009.

[3] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An Architecture-Based approach to Self-Adaptive software," 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.4060

[4] J. Ye, L. Coyle, S. Dobson, and P. Nixon, "Using situation lattices to model and reason about context," *Fourth International Workshop on Modeling and Reasoning in Context*, 2007.

[5] E. Bouix, M. Dalmau, P. Roose, and F. Luthon, "A multimedia oriented component model," *AINA 2005 - The IEEE 19th International Conference on Advanced Information Networking and Applications*, 2005.

[6] L. A. Zadeh, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems*, vol. 100, no. Supplement 1, pp. 9–34, 1999.

[7] P. D. Haghighi, S. Krishnaswamy, A. Zaslavsky, and M. M. Gaber, "Reasoning about context in uncertain pervasive computing environments," in *Proceedings of the 3rd European Conference on Smart Sensing and Context*, ser. EuroSSC '08. Berlin, Heidelberg: Springer-Verlag, 2008, p. 112–125. [Online].

[8] T. Gu, H. K. Pung, D. Q. Zhang, H. K. Pung, and D. Q. Zhang, "A bayesian approach for dealing with uncertain contexts," 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.3509

[9] C. Wojek, K. Nickel, and R. Stiefelhagen, "Activity recognition and Room-Level tracking in an office environment," in *Multisensor Fusion and Integration for Intelligent Systems, 2006 IEEE International Conference on*, 2006, p. 25–30.

[10] K. Hasan, H. A. Rubaiyeat, Y. Lee, and S. Lee, "A reconfigurable HMM for activity recognition," 2008.

[11] P. Diaconis, "[A mathematical theory of evidence. (Glenn shafer)]," *Journal of the American Statistical Association*, vol. 73, no. 363, pp. 677–678, 1978. [Online]. Available: http://www.jstor.org/stable/2286624

[12] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: aiding the development of context-enabled applications," in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, ser. CHI '99. New York, NY, USA: ACM, 1999, pp. 434–441. [Online]. Available: http://doi.acm.org/10.1145/302979.303126

[13] T. Gu, H. K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *J. Netw. Comput. Appl.*, vol. 28, p. 1–18, Jan. 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1053030.1053031

[14] D. Zhang, T. Gu, and X. Wang, "Enabling context-aware smart home with semantic web technologies," *International Journal of Humanfriendly Welfare Robotic Systems*, vol. 6, no. 4, p. 12–20, 2005.

[15] D. Romero, R. Rouvoy, L. Seinturier, S. Chabridon, D. Conan, and N. Pessemier, "Enabling Context-Aware web services: A middleware approach for ubiquitous environments," in *Enabling Context-Aware Web Services: Methods, Architectures, and Technologies*, M. Sheng, J. Yu, and S. Dustdar, Eds. Chapman and Hall/CRC, 2010, pp. 113–135. [Online]. Available: http://hal.inria.fr/inria-00414070/PDF/chapitre.pdf

[16] A. Bouzeghoub, C. Taconet, A. Jarraya, N. Do, and D. Conan, "Complementarity of process-oriented and ontology-based context managers to identify situations," in *ICDIM*, 2010, pp. 222–229.