2005－UBI－8（14）
2005／6／9

A Middleware Architecture for Community Computing with Intelligent Agents

Seungwok Han and Hee Yong Youn

School of Information and Communications Engineering
Sungkyunkwan University, Suwon, Korea
donny@devg.org, youn@ece.skku.ac.kr

ABSTRACT

The applications for ubiquitous system require not only to use the resources distributed in the environment but also to provide intelligent services to the users. In order to satisfy the requirement, the applications need to be developed using intelligent agents providing optimized services to each user. Furthermore, the platform itself needs to be able to support platform-level service optimization with self-growing capability.

In this paper we propose an intelligent middleware architecture displaying high flexibility and scalability with reconfigurable and self-growing elements by adopting a hybrid architecture and dynamically configurable reflective ORB. It consists of two internal layers, one external layer, and various tools forming an efficient agent-based application platform. It also provides a development toolkit, agent management system, and context-oriented interface definition language for context awareness, which are important for developing efficient agent-based applications. The effectiveness of the proposed architecture is demonstrated using an office service scenario.

Keywords: Agent platform, community computing, context oriented, middleware, self-growing.

1. INTRODUCTION

Nowadays, developing large-scale distributed applications has become one of the main tasks for capitalizing the ubiquitous computing environment. The middleware platforms enabling interoperability of diverse components and abstracting from the details of execution environments are increasingly important for software development because ubiquitous systems are getting more and more complex and heterogeneous. The applications developed using the middleware platform are expected to be autonomous, scalable, intelligent, and adaptive to the environments.

Recently, advanced technologies in the wireless communication, mobile computing, intelligent agent, and real-time system, etc. have enabled a new class of applications that require ubiquitous access to the information anywhere and anytime. These distributed applications for ubiquitous system require not only to use the resources distributed in the environment but also to provide intelligent services to the users. In order to satisfy the requirement, the applications for the distributed system need a high degree of flexibility and adaptability in order to deal with heterogeneous platforms and dynamic ubiquitous environments. Also, they need to be developed using intelligent agents providing optimized services to each user. Furthermore, the platform itself needs to be able to support platform-level service optimization with self-growing capability.

In this paper we propose an intelligent middleware architecture displaying high flexibility and scalability with reconfigurable and self-growing elements by adopting a hybrid architecture and dynamically configurable reflective ORB. We describe the operational mechanism of the proposed intelligent middleware architecture using an office service scenario of ubiquitous computing.

The rest of the paper is organized as follows. Section 2 describes the related work and Section 3 presents the proposed middleware architecture for ubiquitous computing. An experiment using an office service scenario is presented in Section 4. Section 5 concludes the paper with some remarks.

2. RELATED WORK

An important requirement of a middleware system in ubiquitous computing environment is a highly configurable and adaptive execution environment that dynamically reacts to the changes in the context and goals. To satisfy the requirements we need an agent platform providing intelligent, reflective, and adaptive services. Furthermore, the platform requires to support the community computing concept that allows optimizing services between the devices and agent groups.

2.1 Agent Platform

Agents can be defined to be autonomous, problem-solving computational entities capable of effective operation in dynamic and open environments [1]. The characteristics of the agents are autonomy, intelligence, mobility, and social

This research was supported by the Ubiquitous Autonomic Computing and Network Project, 21st Century Frontier R&D Program in Korea and the Brain Korea 21 Project in 2005.
Corresponding author : Hee Yong Youn

ability. Additional characteristics are reactivity for reacting to the change of environment, veracity for prohibiting wrong information, and rationality for supporting rational method. The agents can be classified into multi-agent and mobile-agent. The multi-agent is for handling complex operations requiring collaboration between the agents to be completed. The mobile-agent moves itself through the network to process the operations. For this reason mobile agent is used for mobile computing in wireless network.

The agent platforms have to support agent communication language such as ACL and KQML to allow collaboration between the agents residing in different agent platforms. The agents provide designers and developers with a way of structuring an application around autonomous, communicative elements, and lead to the construction of software tools and infrastructure to support the design metaphor [2]. In these days, there exist two common agent platforms, which are JADE [3] and Aglet [4].

2.2 Reflective Middleware

In the reflective model, the middleware is implemented as a collection of components that can be configured at application startup time. The middleware interface is unchanged and can be used by the applications developed for traditional middleware. In addition, the system and application code may also use meta-interfaces to inspect the internal configuration of the middleware and, if needed, reconfigure it to adapt to the changes in the environment. In this manner, it is possible to select networking protocols, security policies, encoding algorithms, and various other mechanisms to optimize the system performance for different contexts and situations.

In general terms, reflective middleware refers to the use of a causally connected self-representation to support inspection and adaptation of the middleware system. Unlike traditional middleware constructed as a monolithic black box, reflective middleware is organized as a group of collaborating components. This organization permits configuration of very small middleware engines that are able to interoperate with traditional middleware.

There exist several different types of projects in this regards such as OpenORB (Lancaster Univ.) [5], 2K Project (UIUC) [6], and so on.

2.3 Adaptive Middleware

The ultimate objective of the adaptive middleware architecture is to control application-aware adaptation behavior and optimize the adaptation strategy towards application-specific performance criteria. In order to accomplish the goals, the middleware architecture consists of adaptors, tuners, configurators, and negotiators. The components cooperatively monitor the application and system states, control the applications to carry out adaptation decisions, and eventually meet the pre-specified performance criteria such as tracking precision.

The major responsibilities of the adaptive middleware architecture are the followings [7].

- The adaptive middleware architecture interacts with the underlying operating system, and accurately observes the current state of the system and application mainly with respect to the resource availability. This feature is implemented in a component referred to as the observation task.

- The middleware architecture needs to decide the adaptation choices and actions to be carried out in the application so that the adaptive behavior can be both stable and fair to other concurrent applications in the same end system, and highly configurable in terms of adaptation agility. The agility represents the sensitivity or responsiveness of the application when adapting itself to external disturbances. These responsibilities of the adaptive middleware architecture are integrated in a component referred to as the adaptation task. Since it depends on accurate observations produced by the observation task, we refer to the combination of both the components as the adaptor.

- In order to balance between globally optimized and fair control decisions and the requirements of meeting diversely different critical performance criteria in different applications, we introduce the tuners and configurators. These components translate the output of the control algorithms in the adaptors into actual parameter-tuning actions or reconfiguration choices to be carried out during the execution of applications.

- In extreme cases, in order to deal with prolonged period of limited resources and degraded qualities, negotiators are activated to coordinate with other end systems. In the case of the tracking application, the negotiators are responsible to locate the new active camera server via the gateway.

2.4 Community Computing Middleware

The goal of PICO [8] is to provide autonomous and persistent services to the users composing dynamic community of software objects which autonomously process the jobs instead of them or their devices. It creates mission-oriented dynamic computing communities that perform tasks for users and devices. It consists of autonomous software entities called *delegents* (or intelligent delegates) and hardware devices called *camileuns* (or connected, adaptive, mobile, intelligent, learned, efficient, ubiquitous nodes). It can represent the interrelation between the *delegent* and *camileun* resources using community operation.

It consists of four layers. First, the *camileun* (physical) layer consists of the hardware, network, operating system and drivers. Second, PICO-compliance software adapts existing hardware devices to the PICO environment with communication module. Third, *delegents* are created for carrying out various tasks and services in the communities. Last layer is community managing.

2.4 Event Service

The suppliers produce events and consumers receive them in the Event Service server. Both suppliers and consumers connect to an event channel or several event channels. An event channel transfers events from suppliers to consumers without requiring suppliers to have information on the

consumer or vice versa. The event channel working as a central mediator in the Event Service server. It is responsible for supplier and consumer registration, clear, timely, and reliable event delivery to all recorded consumers, and the control of errors associated with unresponsive consumers.

The Event Service server provides two models for event transfer: the push and the pull model. On the push model, the suppliers push events to the event channel, and the event channel pushes events to the consumers. Figure 1 shows the push type event delivery. Note that the arrows originated from the client side point the server side.

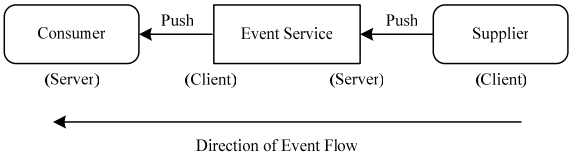

Figure 1. The push model.

For the pull model, the activation causing event flow occurs in the opposite manner: the consumers pull events from the event channel mediator, and the event channel pulls events from theee suppliers. The pull model is shown in Figure 2.

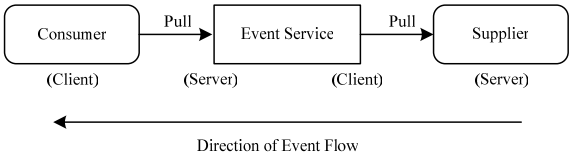

Figure 2. The pull model

Event channels allow multiple suppliers and consumers to be coupled to them. Since some of them may want to use the push model while the others want to use the pull model, event channel supports four different models for event delivery; push/push, push/pull, pull/push, and pull/pull model for supplier and consumer, respectively. These four models differ in the degree of activeness of suppliers and consumers.

3. THE PROPOSED MIDDLEWARE

In ubiquitous environment, the middleware should recognize not only the current situation but also the environment by itself to provide optimal service to the users. It has to provide seamless services to the users regardless of the devices involved using group communication to manipulate the recognized information and reconfigure intelligently. The services need to satisfy the users using intelligent agents. In this section we propose an intelligent middleware architecture satisfying the requirements.

3.1 The Basic Concept

The proposed middleware adopts both the reflective middleware and adaptive middleware concept to construct a flexible platform for the agents and provide development tools. It consists of two internal layers, one external layer, and various tools forming an efficient agent-based application platform as shown in Figure 3. The internal layer consists of the communication platform layer and agent platform layer. The communication platform layer is composed such that it can provide various services based on situation and location using efficient communication protocols with light-weight devices in wire and wireless environment. The agent platform layer is composed of the components so that it can maximize the efficiency of the service, adapt itself to the environment, and accommodate the advantage of diverse agent systems. The external layer is composed of self-growing engine and ontology-based situation-awareness engine for providing intelligent services.

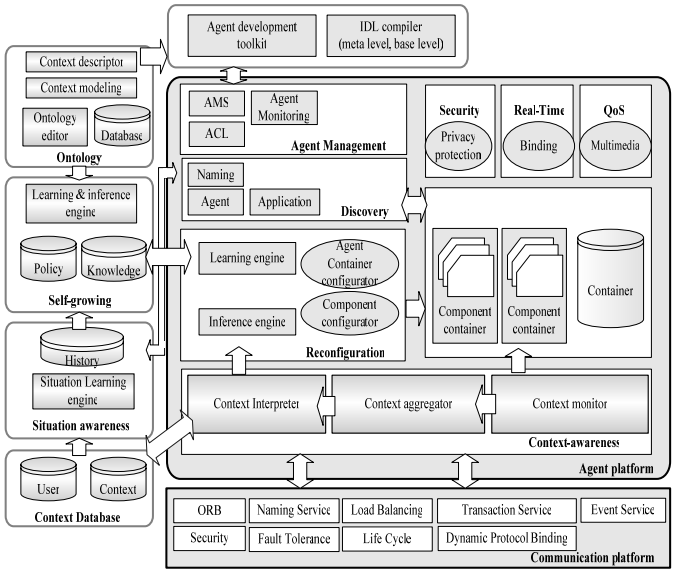

Figure 3. The proposed intelligent middleware architecture.

Additionally, the proposed middleware platform supports context-oriented interface definition language (COIDL) for representing the situation efficiently when developing an agent development toolkit and components supplying situation data.

3.2 The Communication Platform Layer with Adaptor

The communication platform provides event service and context-based naming service to support efficient situation-awareness through ORB, transaction service for assuring consistency and integrity of data and providing recovery method when the data have an error, and dynamic protocol binding to adapt to diverse network environment, load-balancing, security, fault tolerant service, and etc. The communication platform for embedded devices provides MOM(message oriented middleware)-based service. The service is composed using event service, and embedded device can utilize the agents of the intelligent middleware using an adaptor. Channel creation and management of event service of the communication platform are dynamically performed based on required context in the

agent and agent group. Event service is composed so that it can assure rebinding of a protocol and optimized transmission according to the situation of the transmitted data.

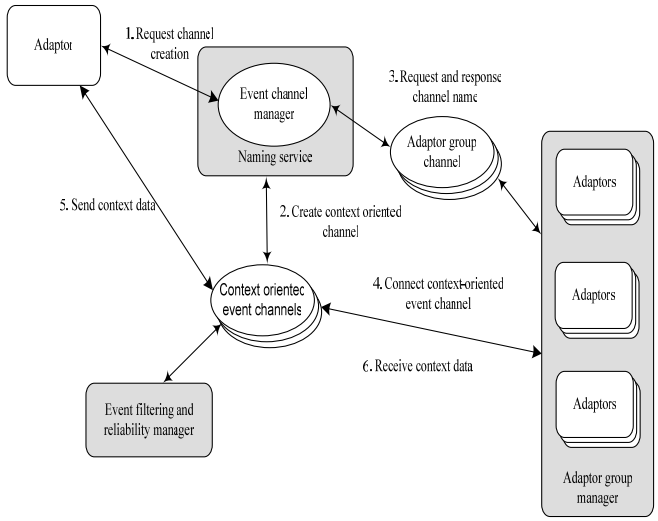

Figure 4. The adaptor and context-oriented channel management.

Each adaptor has a default channel to communicate with other adaptors in the group. The transmitted data is related to the specific conditions of the adaptors. The context information uses context-oriented channel which is managed by event channel manager within the naming service. The context-oriented channel is automatically created, named and deleted by channel messages from the adaptor. This mechanism let the system administrator avoid unnecessary task and waste of the system memory for unused channels. Additionally, the adaptor has its own queue to support reliable message transmission. The queue size is specified by system resources, the size of context and transmission frequency.

The operational mechanism of Figure 4 is as follows. When the adaptor establishes connection with the intelligent middleware, it uses default channel to request channel creation with certain properties, which consist of context and specific adaptor information (Act 1). The default channel is an adaptor group channel, which is managed by platform administrator. The event channel manager creates an event channel with adaptor properties to assign an appropriate name (Act 2). Then the adaptor is treated as a context supplier. The created event channel is monitored by the manager along with the context supplier for dynamic management. If other adaptors request a specific context through the adaptor group channel, the naming service provides the channel name related to the context to connect to a context-oriented event channel (Act 3-4). The context supplier transmits context information through context-oriented event channel and the adaptor group receives the context information (Act 5-6).

The adaptor has three interfaces; initialization, connection, and transmission interface. Iinitialization interface consists of uTAdaptorInit and uTAdaptorUninit method. The uTAdaptorInit method has one argument indicating the communication platform within the domain. The uTAdaptorUninit method disconnects default channel and destroys all the related threads.

Connection interface consists of three proxy methods, which are related to the context-oriented event channel. These methods do not require any arguments to connect any specific channel because all the channels are managed automatically. All the connect methods are for only deciding whether the requesting object is a supplier or consumer. The disconnect method of pull and push proxy is uTProxyDisconnect requiring no argument.

Transmission interface consists of MessageSend and GetMessage method. The MessageSend method is used by suppliers and channels while GetMessage method is used by channels and consumers. The GetMessage returns the message received from the connected channel. Table I summarizes the interfaces and methods of adaptor.

Table I. Adaptor interfaces and methods.

Interfaces	Method
Initialization	int uTAdaptorInit(char* pszChannel);
	int uTAdaptorUninit();
Connection	int uTPushProxyConnect(bool bSupplier = true);
	int uTPullProxyConnect(bool bSupplier = true);
	int uTProxyDisconnect();
Transmission	int uTMessage_Send(char * szMsg);
	char* uTGetMessage();

3.3 The Agent Platform Layer

The agent platform provides component container service so that the components can execute self-creation, self-management, and self-destruction using context-awareness service that passes filtering and compounding context information collected by context monitor. It also contains reconfiguration module which dynamically reconfigures itself by looking up the agents, services, and components. The agent management system (AMS) manages the agents in the community. Moreover, the agent platform provides a reliable and optimized service through real-time, QoS, and security service.

The agent platform supports community computing by dynamically creating and destroying agent groups using the management system and agent discovery system. An optimized service is provided through dynamic creation of interrelated groups according to the agent that requests a service, and exchanges and processes related context among the groups. Priority of the context and agent, related context, device, and service, is applied at this time. The context created by the process is finally confirmed by the user, and if the user is dissatisfied with the decision of the agent, the service is reconfigured. Such approach is employed because it is impossible that the users are always and fully satisfied

with autonomic service provided by an engine lacking context information.

Figure 5 shows the agent execution environment. The community computing concept is supported by the master agent in the master agent container with DF.

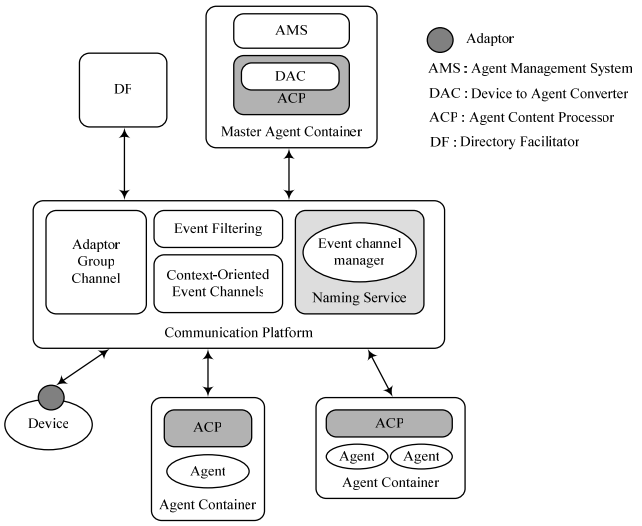

Figure 5. Agent and device management with the adaptor and communication platform.

The agent and adaptor have property and unique ID assigned by the naming service. The property consists of service type, service ID, and service content. The service content is a composition of the context for specific services defining the action of the agent. Each agent has context parsing algorithms abstracting the context of the service content to reduce the load of server process. The ACP is a package of the context parsing algorithms managed by agent container. The DAC allows the devices to be used as an agent. It encodes and decodes adaptor messages into interoperable agent communication messages in the platform. It is important that all the agents and devices are managed by the master agent within master agent container through the DAC component.

The DF supports for constructing agent groups with inter- and intra-domain agents and adaptors. In this process the DF compares the service type of the agents, and the service ID is used to make a relationship among the agents.

All the agents and devices have advertisement mechanism to support discovery. The advertisement messages are periodically generated depending on network connectivity and system condition. Additionally, the master agent requests heartbeat messages to check the system conditions through default channel.

3.4 The Situation-awareness Layer

The situation-awareness layer provides history-based context information collected by various devices in the environment to the services and applications. It analyzes the requirements of the system and services to provide appropriate information. In order to satisfy the requirements, it needs conceptualization, concretization, and structuring of the context to be shared and reused among different domains for inference and learning.

Situation modeling-based ontology and situation-aware components provide high-level situation information to the composite agent-based group community. Also, it makes possible for the communities to effectively communicate with each other by reducing network data. Additionally, the self-growing component supports effective platform reconfiguration based on dynamic policy determined by learning/inference engine modifying static policy for creating a new policy with the history of platform reconfiguration.

3.5 Agent toolkit and Context-Oriented Interface Definition Language (COIDL)

The agent developers should have knowledge on the toolkits, standard specifications, and the agents already developed in the ubiquitous environment. Therefore, at the design level, we provide typical patterns for designing agents, testing tool for performance evaluation, and agent toolkits for automatically managing them at implementation level.

To design and develop the agent that needs to represent special contexts, the type of the contexts that has to be used in the existing platform need to be identified. Moreover, the agent must understand the properties of the context through API's references. At last, the existing programming languages need to define and use the interfaces and functions using Hungarian Notation with ORBA IDL. However, it will be complicated to develop an application with a large number of services. The COIDL can explicitly define the role of the components by defining the prefix, middle, and suffix using the Hungarian Notation and CORBA IDL to efficiently manipulate the contexts.

4. The Operational Mechanism

In this section the operational mechanism of the proposed architecture is demonstrated using a scenario. Here an urgent e-mail arrives at the computer of User-A's office while he/she is going to the office. At this time, the agent on the computer sends the information to the User-A's PDA. (Act. 1) User-A requests the meeting management agent reservation of a meeting room after reading the e-mail. (Act. 2) The meeting management agent reserves a meeting room by adjusting the meeting schedule of the company considering importance of the requested meeting. (Act. 3) After the reservation, a reservation confirmation message is sent to the User-A's PDA. User-A sends a notification mail to the coworkers regarding the urgent meeting after checking the time and place reserved. (Act. 4) The e-mail automatically includes the subjects, starting time, and place of the meeting. When the PCs of the coworkers receive the e-mail, the agents on them send it to their PDA or cellular phone through a similar process shown above. User-A recognizes that the data required in the meeting are at home, and thus searches and finds them in the PC at home. (Act. 5)

The selected data is automatically sent to the computer of the meeting room and printed by the printer agent to supply them to the attendees before the start of the meeting. (Act. 6-7)

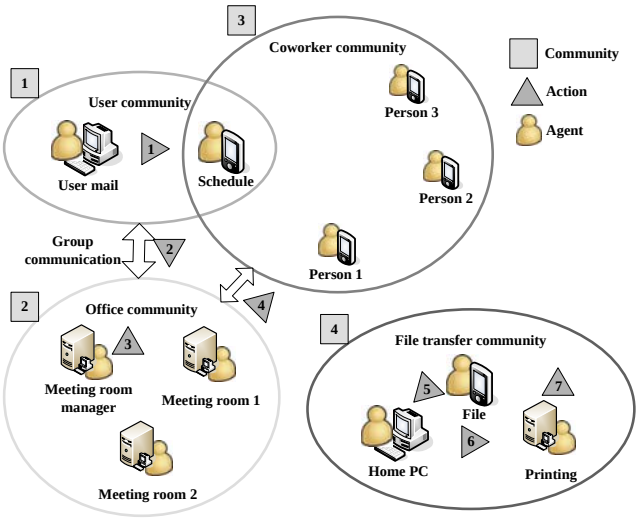

Figure 6. The ubiquitous office service scenario.

The agent discovery service in the proposed middleware organizes community through the search of the agents related to the current context while holding the features of each application. For example, when reserving a meeting room, the system needs the schedule of User-A and the persons in charge of the company, and adjusts all the schedules of the related agents after reservation. (Community. 2,3) At this time, agent group is organized using the relationship between the related agents. The agent group and each agent can efficiently exchange messages each other by sending data through group communication channel and finally adjust the schedules. (Act 2, 4)

5. CONCLUSION AND FUTURE WORK

The importance of intelligent services in ubiquitous computing environment need to revisit the design requirements of future intelligent middleware to cope with the diverse challenges of services over agent-based community computing. In this paper we have presented the overall architecture of the proposed intelligent middleware system. The presented middleware architecture designed to support active deployment of intelligent community service is highly flexible and scalable with reconfigurable and self-growing elements.

The proposed intelligent middleware system is implemented and deployed in CORBA platform with agent related services. The implementation of the intelligent middleware provides us with a unique opportunity to evaluate the system and service, important insights on the complex interaction between agents and communities in heterogeneous computing environment.

Three challenging tasks remain to be addressed as future works. First, practical important applications such as well-being health care in U-city environment and U-office need to be implemented and tested based on our middleware architecture. Second, the proposed architecture needs to be evaluated in terms of the computational overheads in each layer and service. Third, the overall services need to support industrial standard and API reference to implement specific intelligent agent service for the programmers.

REFERENCES

[1] Michael Luck, Peter McBurney, Chris Preist “Agent Technology : Enabling Next Generation Computing” AgentLink community. (2003)

[2] N.R. Jennings. “An agent-based approach for building complex software systems” communications of the ACM, 44(4), 35-41 (2001)

[3] JADE, Java Agent Development framework http://jade.cselt.it

[4] IBM Japan Research Group “Aglets Workbench,” web site: http://www.trl.ibm.com/aglets

[5] Blair, G. S., G. Coulson, et al. "The Design and Implementation of Open ORB version 2"." IEEE Distributed Systems Online Journal2(6), 2001

[6] Fabio Kon, Roy Campbell, M. Dennis Mickunas, Klara Nahrstedt, and Francisco J. Ballesteros. “2K: A Distributed Operating System for Dynamic Heterogeneous Environments.” in 9th IEEE International Symposium on High Performance Distributed Computing. Pittsburgh. August 1-4, 2000

[7] Baochun Li, Won Jeon, William Kalter, Klara Nahrstedt, Jun-Hyuk Seo., "Adaptive Middleware Architecture for a Distributed Omni-Directional Visual Tracking System," in Proceedings of SPIE Multimedia Computing and Networking 2000 (MMCN 2000), pp. 101-112, January 25-27, 2000.

[8] Mohan Kumar, Behrooz A., Shirazi, Sajal K. Das, Byung Y. Sung, David Levine, Mukesh Singhal “PICO : A Middleware Framework for Pervasive Computing”, IEEE pervasive computing magazine Vol 2, Issue 3, pp72-79 (2003)

[9] Fabio Kon, Fábio Costa, Roy Campbell, and Gordon Blair., “The Case for Reflective Middleware.”, Communications of the ACM. Vol. 45, No. 6, pp. 33-38. June, 2002.

[10] Chan, A.T.S., Siu-Nam Chuang. “MobiPADS: a reflective middleware for context-aware mobile computing”, Software Engineering, IEEE Transactions on Volume 29, Issue 12, pp. 1072 – 1085, Dec. 2003.

[11] Object Management Group, “CORBA Components” Version 3.0, formal/02-06-65, June 2002.

[12] Sai-Lai Lo and David Riddoch, “The omniORB version 4.0 User’s Guide”, AT&T Laboratories Cambridge, October , 2004.

[13] M. Henning and S. vinosky, “Advanced CORBA Programming with C++” addition-wesley, Boston, 1999.

[14] Foundation for Intelligent Physical Agents, FIPA Agent Management Specification, SC0023J, 2002