Cogma: A Middleware for Cooperative Smart Appliances for Ad hoc Environment

Nobuo Kawaguchi

Information Technology Center, Nagoya University
1, Furo-cho, Chikusa-ku, Nagoya ,464-8601, Japan, kawaguti@itc.nagoya-u.ac.jp

ABSTRACT

With recent advances in hardware technology, we now have many information appliances which can be connected to a network. However, each device often only serves a limited function and is not designed to integrate with other devices or support different environments. We must establish a framework for network devices that can cooperate with other devices. This study proposes a middleware for the construction of an ad hoc network that does not require a dedicated server and that provides a cooperative application between smart appliances. Our middleware adopts mobile agent technology to enable dynamic software installation. With this feature, we can upgrade software or change the configuration of embedded devices without un-installing them.

Keywords: Ubiquitous Computing, Pervasive Computing, Ad hoc Network, Mobile Agent

1 INTRODUCTION

With recent advances in technology, various information appliances are becoming embedded and used in every day life. In an automobile, for example, a large number of embedded microprocessors are used to control the air conditioning, navigation, and stereo. In a house, the TV set, refrigerator, facsimile, air conditioner, and many other electric appliances have an inbuilt microcomputer chip, however, each appliance is designed to work independently. As a result, you cannot playback music contained in a notebook PC on the car stereo, or transfer location data from your PDA to the car navigation system. Currently, there are many embedded appliances but they do not work in cooperation with each other as each appliance only has the minimum required hardware and software to reduce costs and no middleware is available to integrate these appliances. We must establish a framework for providing network access for devices that will increase in variety, so that we can build an expandable system that can support new and dynamic operating environments.

In this paper, we propose a middleware named “Cogma” that enables the construction of an ad hoc network that connects information appliances and various sensors that are brought to conference rooms, exhibition sites, homes and offices, without requiring a dedicated server, and that creates cooperative applications and requires no presetting.

Cogma (Cooperative Gadgets for Mobile Appliances) is a group of cooperative gadgets or software for mobile information appliances. These “mobile” appliances include those carried by people, such as cellular phones and PDAs, as well as information equipment dynamically deployed in conference rooms and exhibition sites, and semi-fixed information equipment such as “smart” home electric appliances that can be freely moved around within a house. The latter appliances have been designed to sit in a fixed location and various settings are required at the time of installation. Homes now have many such appliances and each requires specialized knowledge for its initial setup. Even if the settings of an individual appliance are simple, there are many appliances requiring setup. Our purpose of the study is to minimize such a humble operation for appliances.

Our middleware has adopted mobile software technology for dynamic code installation and using this feature, it is easy to upgrade software or change the configuration of an embedded appliance after it is installed. It is also easy to build a cooperative application by connecting various appliances that do not require any individual manual settings. One of these application is a smart conference room with a projector and a screen, lighting and audio equipment, and a group of PCs linked together. Another example is a smart living room with temperature and photo sensors, air conditioning, lighting, a TV set, and a stereo that function in harmony. The middleware is built on an ad hoc network technology, and the adoption of dynamic software makes it flexible and thus it can adapt to various different environments.

In the following, we present the components for the software system that are required for connecting embedded appliances in Section 2, and discuss the design and implementation of our middleware in Section 3 and the subsequent sections.

2 REQUIREMENTS FOR DYNAMIC COOPERATION BETWEEN EMBEDDED APPLIANCES

In this study, we have assumed that the embedded appliances under consideration have network communication capability. We also assumed that these devices are able to detect a link connection or disconnection to other nodes, and thus they can form an ad hoc network. We consider that the following characteristics are required for dynamic cooperation between the various embedded devices.

(A) Self Understanding of Basic Functions

Generally the information necessary for the operation of an individual appliance is only known by that appliance. It is required that each appliance stores the information necessary for its own functions, and provides it when required by another system. In addition, each appliance should be fully operational with this information alone.

(B) Dynamic Modification and Addition of Functions

Once an embedded appliance is installed, it is not a simple task to re-install software. Many appliances support firmware upgrades but the use of a dedicated PC and software, or at minimum rebooting, is required. It is required that the basic software can be modified and added to through the network while the system is operational.

(C) External Controllability

For embedded appliances with networking capability, it is useful if their information can be externally accessed, and they can be controlled via a remote system. This feature enables embedded appliances that have no physical user interface to be integrated with other appliances through a remote control device.

(D) Operable with Limited Resources

Generally, embedded appliances have only limited memory and CPU power. For such appliances to remain operable, it is required that the minimum set of functions and other functions be implemented separately.

Embedded appliances with the above features can operate cooperatively by dynamically recognizing each other's functions. External monitoring is also possible with feature (C), however, considering the hardware limitations of embedded appliances, implementation should be as simple as possible.

In this study, we also assume that the granularity of state changes in each appliance is approximately once in five seconds. A major reason for this is that network and other environmental changes are detected by polling.

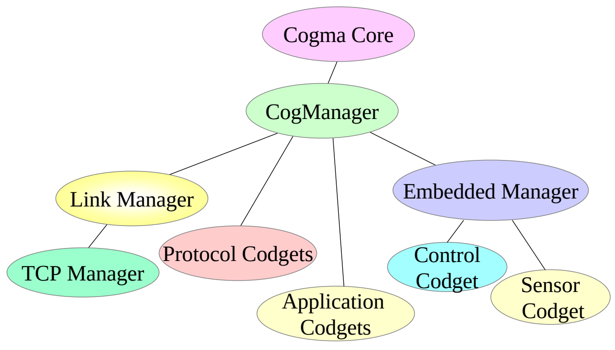


Figure1: Module Hierarchy

3 DESIGN OF MIDDLEWARE TO FACILITATE COOPERATION AMONG EMBEDDED APPLIANCES

This section describes the design of Cogma (Cooperative Gadgets for Mobile Application), which is middleware to facilitate cooperation among embedded devices. To satisfy the above requirements, we employed mobile software technology in Cogma. We satisfy requirement (B) by designing most components to be transportable. Therefore, we designed the system in the hierarchical form shown in Figure 1. In our middleware, the basic units are transportable components called 'codgets' (Cooperative Gadgets). We call each device which runs the middleware a 'node'. Each codget can move over the communication links between the nodes. For requirement (A), we developed an 'Embedded Manager' to manage the resources of the appliances. The Embedded Manager is also a codget, and thus it is managed by 'CogManager' which manages all codgets. Requirement (C) is realized by employing a codget that acts as an interface between the external node and its internal codgets. Finally, for requirement (D), we construct each codget to be as small as possible. For example, in the CogManager, we separate the management mechanism from the graphical user interface and implement each of them in a different class of codget. With this separation, embedded appliances which have no physical user interface can reduce memory consumption. For the operation of small memory embedded devices, we adopt Personal Java 1.1.3.

In the following, we explain each component of Cogma.

- **Base System (Cogma Core)**

  This module manages the dynamic transportation of software. All other modules can be modified or exchanged while this system is running.
- **Codget Manager (CogManager)**

  This module manages each codget and controls the registration and deregistration of codgets. The module also plays an important role as the interface between the link manager and the codgets.
- **Communication Link Manager (Link Manager)**

  This codget monitors each communication link, and notifies the CogManager of a 'connection' or 'disconnection' of a link. This codget also sends and receives codgets over the links.
- **Embedded Device Manager (Embedded Manager)**

  This codget controls the modules that are inherent to each devices such as sensors and switches. Embedded managers in the same network exchange information with each other.
- **Protocol Codgets**

  These codgets provide protocols for communication between codgets or between nodes.
- **Application Codgets**

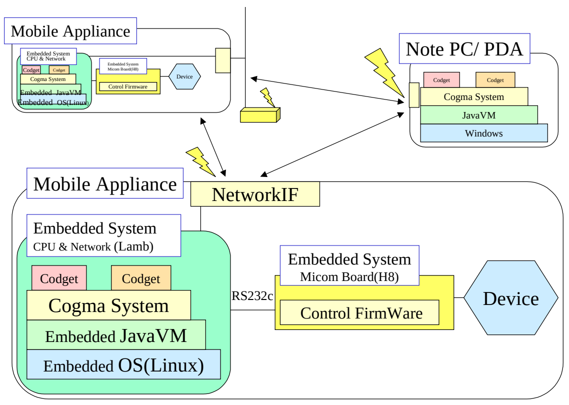


Figure2: Relationship among Devices.

These codgets implement individual cooperative applications.

Since modules under the CogManager are implemented as a codget, they can be dynamically modified or exchanged during run-time. By using a protocol codget, it is possible to dynamically deploy a new network protocol that is adapted to the current environment or situation. As illustrated in Figure 2, Cogma works on a variety of devices including notebook PCs and embedded devices.

3.1 Basic Module Unit: Codget

Our middleware is designed to use software modules called codgets. A codget corresponds to a mobile agent in a mobile agent system. However, a codget is not necessarily to move because it is sometimes used to manage resources, provide a protocol, and implement a manager. Though, a codget enables the dynamic system updates[5]. Table 1 lists the basic APIs for codgets. A codget can be stopped by external control, and can be made permanent by serializing. When a Codget is moved, it does not always carry the program code. The code is sent on demand to reduce the transmission overhead. Each codget has a unique CodgetID that corresponds to the initiating host. Since the CodgetID never changes, even when the codget is moved or serialized, it is used in communication between codgets as will be described in the next section.

3.1.1 Communication between Codgets

To implement the middleware and application as simply as possible, Cogma has no dedicated method such as messaging systems for communication between hosts or between codgets. Instead of using a messaging system, Cogma adopts a unique solution to enable cooperation between codgets. In this method, when a codget that has the same CodgetID is registered at the same node (CogManager), the onSame method in the existing codget is called with the newly registered codget as an argument. Thus, the existing codget can directly access the stored data of the newly registered codget. Because of this feature, no two codgets within a node can have the same ID. This feature also simplifies the programming of the distributed system.

We consider here a codget that collects information between nodes. This codget, once it has been registered locally, moves to another node while leaving a copy of itself at the local node by calling the “copyTo” API on CogManager. After collecting information at another node, the codget returns to the local node. Then the onSame method of the copy at the local node is called to obtain the necessary data from the returned codget. By using this method, synchronization of information between two codgets can be performed easily. Additionally, communication within the same node and communication between different nodes can be performed using a very similar procedure.

3.1.2 Protocol Codget

Communication between codgets that do not know each other is implemented with a codget called ‘Protocol Codget’. Unlike a regular codget that has a unique CodgetID for each initiation, CodgetID of a protocol codget corresponds to a protocol itself, similar to a TCP port number. In general, to implement a standardized protocol, it is necessary to write a program that conforms to the protocol specifications. By using our method, a protocol codget, rather than a protocol specification, is distributed as a library so a user can implement a protocol simply by calling the library.

Protocol codget is also a codget. Therefore, when identical protocol codgets arrive at a node, ‘onSame’ method of the existing protocol codget is called with the newly arrived protocol codget as an argument. Usually, several codgets executing a same program code can exist on the same node, but several protocol codgets are not allowed to exist on the same node because identical protocol codgets have identical CodgetIDs. Regular codgets have different CodgetIDs if they are initiated at different nodes or times. This feature facilitates the implementation of cooperative applications.

3.2 Codget Management

The CogManager, which manages codgets, also controls managers such as the LinkManager and EmbeddedManager. They will be described in the following sections. The methods ‘register’ and ‘unRegister’ enable each codget to register or unregister an another codget. By using them, codgets can form a group with a hierarchical structure. The method ‘registerLinkMonitor’ registers the codget itself as a LinkMonitor that receives the notification from the LinkManager. When the LinkManager sends notification of the arrival of a new node, the ‘onNew’ method on the LinkMonitor is called. Also, when the LinkManager sends notification of the disappearance of a node, the ‘onBye’ method on the LinkMonitor is called.

By using the method ‘copyTo’ on the CogManager, a codget can move to an adjacent node. In its basic configuration, Cogma can only copy a codget to the adjacent node. To move a codget over the network, a network codget is required to distribute the codget.

3.3 Communication Link Management

Cogma supports various communication links including wired Ethernet, IEEE802.11b, serial, and infrared(IrDA). The LinkManager controls the link, notes any state changes of a link and transports codgets over it. The LinkManager only controls a single hop of communication. For multi-hop communication, a codget that implements a network protocol is used as a Flooding Codget as described above. We have already proposed the ad hoc protocols with MAGNET[1]

When there is a change in a communication link due to the arrival of a new node or the movement of an existing node, the LinkManager notifies any codgets that demand to be informed of the state change(Such codgets are called LinkMonitors). Each Codget operates independently in response to changes in a link. For example, a codget that configures the network based on the link state, performs a network recalculation.

3.4 Embedded Manager

To support the self-understanding of each node, we develop the “Embedded Manager”. EmbeddedManager manage the various sensors, and control devices such as temperature sensor, photo sensor, light controller, etc. First, we define the abstraction of the sensor and control devices. For sensor devices, we assume that there are only three types of sensors, the (1) On-Off sensor, (2) Scalar Sensor and (3) Vector Sensor. Then, for each sensor, we define the following attributes: (A)Sensor Type, (B) Max Value, (C)Min Value, (D) Metric, (E)Resolution, (F)Sensor Value, and (G) Polling interval. For control devices, currently, we assume there are 10 types of functions: (1)On-Off, (2)Up-Down, (3)Pause-Go, (4)Stop, (5)Record, (6)Play, (7)Search, (8)Input-change, (9)Output-change, and (10)Direction-control. The controlling device is also named according to the media it controls, for example, “Light”, “Sound”, “Movement”, “Image”, and so on.

By the abstraction of each device, we can develop embedded managers to manage the sensor and control devices. The embedded manager should have the ability to register and discover each device. In our implementation, the embedded manager on each node exchanges information about its own sensor and control devices.

When an application wants to use a specific device, the application request the embedded manager to locate the spec-object in the abstract form described above using a protocol codget named “LocalEmbProto”. This means that the application implements the “EmbeddedMonitor” interface which monitors the arrival of new devices. If the embedded manager identifies a device that matches the specifications, the manager returns the matched node information to the application. Otherwise, the embedded manager asks other nodes about the specified device using an “EmbProto” protocol codget.

3.5 Security and Group Management

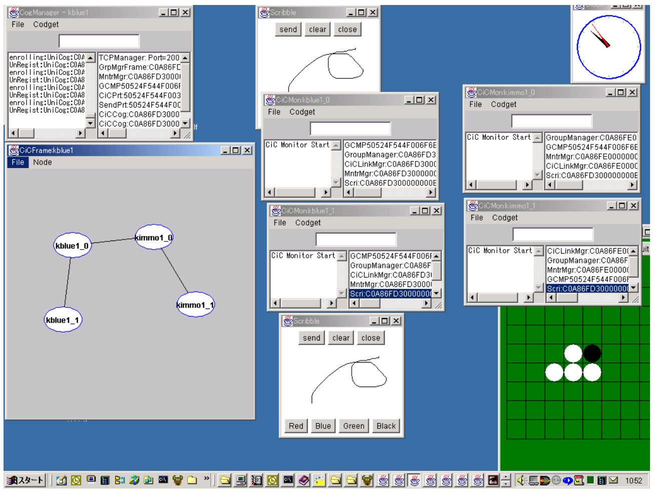
A system that can be controlled externally via a network is required to have a high level of security. Cogma has an authorization control system based on group management[4] where each node belongs to a group that has several authorization levels. A codget that is transported within a group that has management authorization has the privilege of operating the system, and is thus different from other codgets. This feature prevents malicious attacks from a system that does not belong to one of the appropriate groups. Each Cogma system has a group management mechanism for obtaining the group participation state of each node. Since the group definition itself is implemented with mobile software, it is possible to create a variety of groups depending on their status, for example, groups for link state, location, and time. For group authentication in an ad hoc environment, we use a digital signed file to validate the code. In this method, belonging to a group authorizes participation in that group. We are also considering the use of tamperproof hardware[6].

4 HARDWARE ENVIRONMENT

It is difficult to build a CPU into all embedded appliances, and therefore, to control appliances, Cogma uses a microprocessor system that has multiple IO ports. We chose a 16-bit H8/3664 microprocessor for this purpose. For the controller CPU, we used the LAMB-EM-01 (5x86, 133MHz) equipped with a PCMCIA wireless LAN card. These devices are connected with an RS-232C cable (Figure 2). The H8 microprocessor supports a universal infrared remote controller that can control many types of domestic electric appliances. The LAMB-EM-01 runs on Linux and Personal Java 1.1.3, and since Cogma operates on PCs and PDAs (Sharp Zaurus), our prototype system employs cooperative software in order to communicate between these devices.

Unlike notebook PCs that have a powerful CPU and a large memory, the Cogma system in an embedded device operates with limited resources. It implements the minimum necessary components without screen display functions or graphical user interfaces.

Figure 3 shows an operation screen of the current Cogma system. On this screen, Cogma is running on two nodes and two CiCs(Node Monitor) are running on each node.



```
enrolling:UniCog:C0A
UnRegist:UniCog:C0A8
enrolling:UniCog:C0A
UnRegist:UniCog:C0A8
enrolling:UniCog:C0A
UnRegist:UniCog:C0A8
enrolling:UniCog:C0A
UnRegist:UniCog:C0A8
```

```
TCPManager: Port=200
GrpMgrFrame:C0A86FD
MntrMgr:C0A86FD3000
GCMP50524F544F006F
CiCPrt:50524F544F003
SendPrt:50524F544F00
CiCCog:C0A86FD3000
CiCCog:C0A86FD3000
```

```
CiC Monitor Start
```

```
GCMP50524F544F006F6E
GroupManager:C0A86FD3
CiCLinkMgr:C0A86FD3000
MntrMgr:C0A86FD3000000
Scri:C0A86FD300000000E
```

```
GCMP50524F544F006F
GroupManager:C0A86F
CiCLinkMgr:C0A86FD3
MntrMgr:C0A86FD3000
Scri:C0A86FD30000000
```

```
GroupManager:C0A86FE0
GCMP50524F544F006F6E
MntrMgr:C0A86FE000000
CiCLinkMgr:C0A86FE000
Scri:C0A86FD300000000E
```

```
CiCLinkMgr:C0A86FE0
GroupManager:C0A86F
MntrMgr:C0A86FE0000
GCMP50524F544F006F
Scri:C0A86FD3000000
```

Figure 3:Cogma Operating Screen

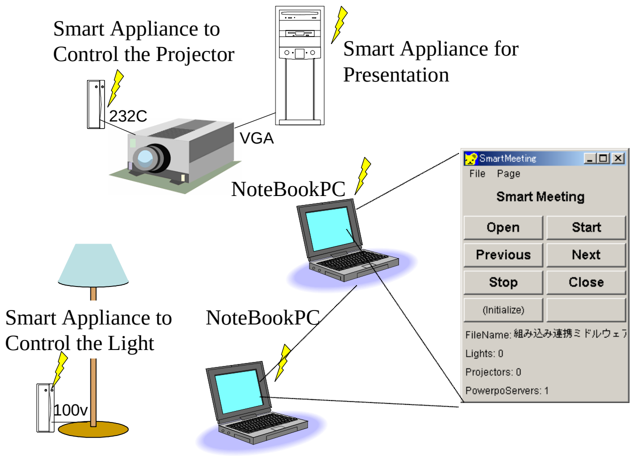


Figure 4: Overview of SmartMeeting

Implemented Prototype Application and Experience

We have implemented several applications which utilize our middleware Cogma such as the control of a fan, a thermo sensor, and so on. One of them is the "SmartMeeting" system shown in Figure 4. SmartMeeting supports a user in using the meeting room devices such as the projector, light, and presentation system, in an ad hoc manner. This means that the user is not required to manually ser up each device to work, even if the user is using to this meeting room for the first time. Each device in the room is equipped with a Smart Appliance which is run by Cogma. When a user enters the room, the user's Notebook PC initiates an ad-hoc network among the appliances in the room using the wireless network. Then, the SmartMeeting control panel lists the devices in the room on the notebook PC screen as shown on the right-hand side of Figure 4. When the user wants to start the presentation, the user simply "Opens" the presentation file, then pushes the "Start" button. In response, the light will turn off, the projector will turn on, and the presentation will start. The user can control the presentation from his own Notebook PC using the SmartMeeting control panel.

We implemented this application using an Embedded Manager, a SmartMeeting control panel codget, and three embedded codgets for each smart appliance. On the basis of this experience, we confirm that it is very easy to develop such applications based on our Embedded Manager. We simply created three controlling codgets and wrote their specifications. Then we coded the control panel with the required specifications. Communication between nodes is facilitated by Embedded Manager, thus the control panel only needs to ask the local Embedded Manager about the specified devices.

5 RELATED RESEARCH

There are a number of current research projects which share a similar framework, which include Jini[7], and Hive[8], however, these technologies do not support ad hoc networks or multilinks because they use Java RMI. With Jini, the users and the service providers are clearly separated by the Lookup server, and thus it differs from Cogma which enables the dynamic transportation of codgets to each node.

STONE[9] is an example of a dynamic cooperative system. STONE is based on finding dynamic services, since it is a framework that can combine services. However, the functional units and service resolver are independent, which is differs from Cogma whose components have the same configuration.

SONA[11] of the Smart Space Lab[12] is targeted at achieving autonomous cooperation among home appliances. It uses Jini for the lookup service that detects appliances and Aglets[14] for the dynamic installation of software. Aglets has no detection function, but the adoption of Jini seems to support a dynamic environment. In the Smart Space Lab, there are a number of appliances are installed, however, the system does not work in ad hoc environments such as another room because the system components are not in same configuration.

MAGNET[1] provided the basis for building Cogma and the experience gained from it was used in the development of Cogma. The greatest difference between Cogma and MAGNET is that the MAGNET design assumes a component similar to a notebook PC that has a screen, network, keyboard and mouse, whereas Cogma does not necessarily require such elements. This feature influences many aspects of the system design. MAGNET used the latest JDK available at the time of its development, while Cogma uses Personal Java for use in embedded devices, although this is beginning to be outdated. With MAGNET we adopt the use of an emulation environment where the appliances are controlled by a "God" who can monitor all communication. In the emulation environment, rebooting is required for reconnection and reconfiguration, however, for embedded devices, rebooting should be keep to a minimum. However, for distributed software development, the monitoring of communication is a required function. The CiC function that monitors software operations in embedded appliances can be used as a platform for efficient Cogma software development.

6 SUMMARY

We have proposed and implemented Cogma, which is a middleware for building cooperative applications that operate between embedded devices. Based on mobile software technology and the experience gained from the development of MAGNET[1], Cogma is a system with dedicated functions for embedded appliances. It can realize a dynamic application within a simple framework. We have successfully built the SmartMeeting application to exemplify the usefulness of our system. Based on this experience, we confirm that the Embedded Manager simplifies the development of the cooperative application. The simple communication framework of the Codget also contributes to the easy development of distributed applications.

We are also considering the application of Cogma to home appliances that have already been installed [13]. For documents and the current state of development of Cogma, please visit the Cogma Project Web site [15].

REFERENCES

[1] Nobuo Kawaguchi, Katsuhiko Toyama, and Yasuyoshi Inagaki, MAGNET: Ad-Hoc Network System based on Mobile Agents , Special Issue for Mobile Agents for Telecommunication Applications , Computer Communication, Vol.23, pp.761--768(2000).

[2] Nobuo Kawaguchi, Yasuyoshi Inagaki, Multi-Link Ad-Hoc Communication System based on Mobile Agents,In Proceedings of the ACIS 2nd International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing(SNPD'01), pp.311--318(2001).

[3] Shunichi Sugiura, Nobuo Kawaguchi, Katsuhiko Tonoyama, and Yasuyoshi Inagaki: Proposal of a File Management System Using Mobile Agents, IPSJ, DiCoMo 2000, pp.145-150 (2000). (In Japanese)

[4] Yuuki Miyagoshi, Nobuo Kawaguchi, and Yasuyoshi Inagaki: Flexible Group Management on Ad Hoc Network Using Mobile Agents, in Proceedings of the 4th International Symposium on Wireless Personal Multimedia Communications(WPMC2001), pp. 835--840(2001).

[5] Nobuo Kawaguchi, Katsuhiko Tonoyama, and Yasuyoshi Inagaki: Dynamically Expandable Software System Based on Mobile Agents, IPSJ, Summer Project Symposium Reports, pp.71-78 (1999).(In Japanese)

[6] Hiromi Haruki, Nobuo Kawaguchi, and Yasuyoshi Inagaki: Mobile Agent Protection Method Using Tamperproof Hardware, IPSJ, DiCoMo2001, pp.43-48 (2001). (In Japanese)

[7] Sun Microsystems. Inc. :`Jini Architecture Specification Version 1.1,' 'http://java.sun.com/jini/specs/(2000).

[8] Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krikorian , and Pattie Maes,``Hive: Distributed Agents for Networking Things," ASA/MA'99(1999).

[9] Masaki Minami, Hiroyuki Morikawa, and Yuki Aoyama: Design of a Network Service Synthesizer, Technical Report of IEICE, IN2000-192, pp.1-8 (2001).

[10] Tomoko Itao, Tetsuya Nakamura, Masato Matsuo, Tatsuya Suda, Tomonori Aoyama, "The Model and Design of Cooperative Interaction for Service Composition," IPSJ, DiCoMo2001, pp.7-12 (2001). (In Japanese)

[11] Muneyuki Aoki, Jin Nakazawa, and Hideyuki Tokuda: Building an Autonomous Information Appliance Control Mechanism, IPSJ, Information Appliance Computing Research Group 1st Workshop, (2001). (In Japanese)

[12] T. Okoshi, S. Wakayama, Y. Sugita, S.Aoki, T. Iwamato, J.Nakazawa, T. Nagata, D. Furukusa, M. Iwai, A. Kusumoto, N. Harashima, J.Yura, N. Nishio , Y. Tobe, Y. Ikeda and H. Tokuda, ``Smart Space Laboratory Project: Toward the Next Generation Computing Environment", in Proceeding of IEEE Third Workshop on Networked Appliances(IWNA2001),(2001).

[13] Yasuo Tan: Home Network Using Installed Home Electric Appliances, IPSJ, Information Appliance Computing Research Group 1st Workshop, (2001).(In Japanese)

[14] Danny B. Lange, Mitsuru Ohshima, Programming and Deploying Java Mobile Agents with Aglets, Addison Wesley(1998).

[15] Cogma Project Web Page:
http://www.cogma.org/