Multi-Layered Architecture of Middleware for Ubiquitous Robot

In-Bae Jeong, Jong-Hwan Kim
Department of Electrical Engineering and Computer Science
KAIST
Daejeon, Republic of Korea
{ibjeong,johkim}@rit.kaist.ac.kr

***Abstract*—This paper proposes a multi-layered architecture of middleware for ubiquitous robots. Ubiquitous robots consist of different platforms with various functions and interfaces. Without a middleware, software agents have to hold information of all ubiquitous robots in advance to access other sensors or mobile robots. This decreases modularity and scalability of the entire system, therefore makes it difficult to develop and maintain software agents. The proposed architecture uses context information to decouple hardware and software agents as a bridge between them. Providing context-awareness to the middleware, software agents is able to get context information without accessing physical sensors directly, so that modularity and scalability are increased. Not only providing context information to software agents, the middleware uses mobile robots to gather as much sensor information as possible to generate context information actively. A middleware architecture is proposed with five layers classified due to device/environment dependencies, which are physical layer, device management layer, context provider and mobile robot(Mobot) scheduler layer, software robot(Sobot) management layer and software agent layer. The proposed middleware is implemented and simulated with virtual sensors in the virtual environment.**

***Index Terms*—Ubiquitous robot, middleware, context information**

I. Introduction

Ubiquitous computing, which was coined by Mark Weiser, motivated developments in computer and network technology with the paradigm shift [1], [2]. It also motivated ubiquitous robotics, the third generation of robotics, following the industrial robot and the personal robot [3], [4]. With developments in computer and network technology, ubiquitous robots will offer desired services by any IT device at any place and time through user interactions and seamless application.

Researches of former generations of robotics were mainly about stand-alone robots for specialized purposes. However, researches of ubiquitous robotics include network-based robot, software agent, multi-robot cooperation as well as human-robot interaction. Internet-based robot control was introduced[5] and Luo et al. proposed the network controlled robot systems to solve internet latency and local intelligence of robots[6]. Ha et al. proposed Service-oriented Ubiquitous Robotic Framework where each sensors and robots are organized into semantic web service[7]. Using knowledge repository, the system understand the user's request and offer a proper service by composing web services. Main focus of researches of ubiquitous robotics is integrating sensors and robots. Multi-robot systems usually are composed of several robots having different architectures, communication interfaces and protocols, so there have been studies about middleware to integrate them. The Object Management Group, Inc. defined Common Object Request Broker Architecture(CORBA), which uses an interface definition language to provide interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems[8], [9].

Although middleware offers common interfaces to access devices which have a variety of communication interfaces and protocols, software agents still should have specific information of sensors and mobile robots to obtain sensor information and control them, respectively. Therefore, when a software agent is developed, those information have to be predefined in the software agent. It causes strong coupling between hardware components and software components. If those are strongly coupled, software agents must be rewritten whenever a new type of sensor or mobile robot is added. As modularity decreases, developing and maintaining software agents get more difficult, and scalability and extensibility also decrease.

To solve these problems, this paper proposed a multi-layered architecture which separates context generation and software agents. If sufficient context information is supplied to software agents, they can decide what to do based on the given context information without accessing hardware devices directly.

II. Preliminaries

A. U-space and Ubiquitous Robot

Ubiquitous space(U-space) is an environment in which ubiquitous computing is realized and every device is networked. Ubiquitous robot(Ubibot) means every robot which is developed in and exists within U-space. All Ubibots are interconnected in U-space with ubiquitous network and provide services to users at any place and anytime. Since ubibots are interconnected through network, the network delay is inevitable and the delay problem reduces the performance of real time applications. However, it can be reduced with limiting the size of U-space.

In order to provide services to users with Ubibots, networked cooperative multi-robot system is needed. In the future, software robot will take the essential role controlling hardware robot interacting with the other robots.

1-4244-2384-2/08/$20.00 © 2008 IEEE

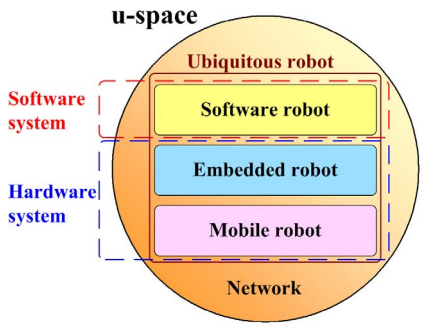


Fig. 1. Ubibots in ubiquitous space

While personal robots need user's explicit command to provide services, Ubibots will decide which service is needed intelligently and provide proper services to users continuously.

Ubibots are classified as software robot(Sobot), embedded robot(Embot) and mobile robot(Mobot) according to their functions and roles in U-space. Sobot is a robot which consists of software and controls other Sobots, Embots and Mobots. Embot is embedded in U-space or Mobots, detects situational information and provides those information to other Ubibots. Mobot has mobility and is capable of providing physical services to users. Ubibots in U-space are shown in Fig.1.

B. Software Robot: Sobot

Sobot is a robot which consists of software. It has context-awareness and provides continuous services interacting with users automatically and calmly. Sobot moves through ubiquitous network into any device in U-space anytime at any place. Sobot has three main characteristics; self-learning, context-aware intelligence and calm and seamless interaction. Sobot takes the main role of intelligence of Ubibots. Sobot recognizes current situation, decides which behavior should be performed and execute it without directly consulting the user.

C. Embedded Robot: Embot

The main role of Embot is sensing and understanding environmental situation. It is characterized by calm sensing, information processing and ubiquitous communication. Embot is implanted in the environment or in Mobots to sense calmly and collect information. Embots also analyze the information about human behavior, status, relationship and environmental conditions such as weather, temperature, etc.

D. Mobile Robot: Mobot

Mobot offers a broad range of services for users and specific functions within a specific U-space. Key properties of Mobot are manipulation and mobility, which can be implemented in various types, such as wheel-types or biped. Mobot interacts with Sobot to provide practical services based on commands given by Sobot, and realizes a broad range of services, such as personal, public, or field services. Mobot uses the implanted Embots as local sensors without interacting the Ubi-server, therefore it also acts as a standalone system having real time applications.

E. Middleware

Middleware allows communication within and among Ubibots which are developed on a variety of platforms using a variety of network interfaces and protocols. Middleware offers common interfaces to Ubibots, thereby making it convenient to componentize the devices and to manage them. It enables the system to make an offer of service irrespective of the operating system, geometric location and type of interface. Since Ubibot system is a cooperative multi-robot system composed of heterogeneous robots, developing middleware is essential, which acts as an impartial link between applications and allows for transparent connectivity between heterogeneous protocols and networks.

III. Multi-layered Architecture for Ubibot

Ubibots in U-space should be interconnected. Since Ubibots are developed on diverse architectures and platforms with different communication protocols, their properties may differ in many ways. Therefore, software agents should have all hardware information of the other Ubibots in advance, but it is very difficult practically and it is rarely possible to expect modularity and scalability.

As Embots and Mobots operate in a real environment, they are dependent on device and environment, while software agents are independent of device and environment as they decide services to provide with the given situational information. Generating contexts, which are obtained from sensor information, is obviously dependent on device and environment. However, since context information itself is a description of a certain situational information, it is independent of device and environment. As mentioned before, strong coupling makes it difficult to organize a scalable and extensible system. Using the contexts as a link between hardware components and software components, they they can be decoupled.

Context does not simply mean sensor information, it means information describing the properties of situations, i.e, a specific situation related with place, object or time. It includes user context, physical context and time context. These contexts are also used to infer the other contexts. Contexts can be obtained directly from sensory information or generated with inferring from the other contexts.

In usual case, software agents generate context but context generation mainly depends on sensor information and environmental information. In order to achieve independency from device/environment, it is proposed that middleware converts sensory information to contexts. Since middleware provides context information, software agents do not need to access physical devices directly. Being independent from device/environment, software agents can easily move to other environment and continuously provide proper services even if the environment changes.

The proposed architecture is composed of five layers: Software agent layer, Sobot management layer, Context

provider/Task scheduler layer, Device management layer and Physical layer as shown in Fig.2. Embots and Mobots in 'Physical layer' physically operate within the ubiquitous environment, where Sobots in 'Software agent layer' transfer into various Mobots and decide services to provide. The proposed middleware consists of 'Sobot management layer', 'Context provider/Task scheduler layer' and 'Device management layer', which is designed to operate in the ubiquitous space.

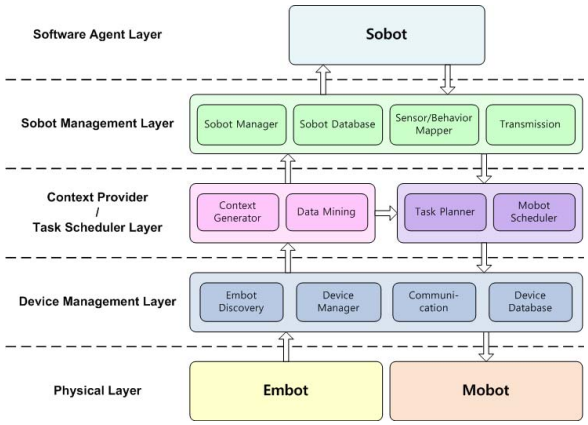


Fig. 2. The proposed multi-layered architecture

A. Physical layer

'Physical layer' includes Embots and Mobots. Usually, raw sensor data including vision data and sound data are large in size compared to network bandwidth. Embot processes the data to generate meaningful information which is used to decide current context.

Embots and Mobots are developed with sensors and actuators, respectively, for their own roles. Embots and Mobots send their device descriptions to device manager to register themselves. The device manager manages devices based on the device descriptions, which are composed of the hardware properties including the type of sensor, the number of sensors, parameter of sensor, location information, the list of available actions, etc.

For platform independency and reprogrammability, Embots and Mobots embed a script engine and libraries to execute any given procedure.

B. Device management layer

Device manager stores the given device descriptions from Embots and Mobots in a device database and manages Embots and Mobots with the database. It monitors ubiquitous network and requests Embot and Mobots to register when they are connected to the ubiquitous network.

Since device manager also provides common interfaces to access Embots and Mobots, it is possible to access them without knowing their hardware-specific information including communication protocols.

C. Context provider/Task scheduler layer

'Context provider layer' and 'Task scheduler layer' are separated in the same level according to their functions.

1) Context provider layer: Context provider layer deduces context information from sensor information. When sensor information are not enough to decide the current context in the environment, it sends a locomotion commands to task scheduler so that Mobots having proper sensors move around to gather more sensor information.

2) Task scheduler layer: Mobots should perform user commands, Sobot commands and locomotion commands from context provider avoiding obstacles and considering the feasibility of the given missions. The main role of task scheduler is to plan tasks based on the priority of tasks and available Mobots.

D. Sobot management layer

'Sobot management layer' stores Sobot information and the types of context information which are needed by Sobots. When the context information are given from context provider, Sobot manger sends the context information to the Sobots. It also manages the transmission of Sobots when they requested to move into other Mobots or virtual environment. Since Sobot is an artificial creature having virtual sensors and virtual behaviors, the Sobot manager should provide appropriate mapping relations to map virtual sensors/behaviors to physical sensors/behaviors of Mobot.

E. Software agent layer

Sobot in 'software agent layer' analyzes the given context information from the Sobot manager and decides which services should be provided to users. While Sobot usually varies depending upon purposes and functions, it basically decides services to provide based on context information.

F. Overall sequence

Fig.3 shows the sequence of Ubibot registration and context generation.

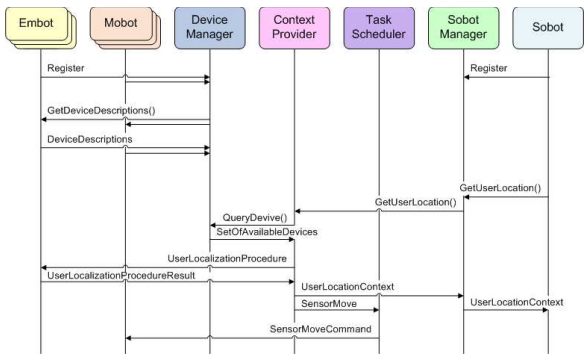


Fig. 3. Sequence diagram

1) Embot/Mobot/Sobot Registration: Embots and Mobots register themselves to device manager and Sobots to Sobot manager(Register). Device manager requests device descriptions to Embots and Mobots(GetDeviceDescriptions()). Embots and Mobots send device descriptions to the device manager(DeviceDescriptions).

2) Context Information Request: Sobots request context information which is needed to Sobot manager(GetUserLocation()). The Sobot manager requests context information to the context provider(GetUserLocation()).

3) Context Information Generation: The context provider requests the list of available devices to the device manager(QueryDevice()). The device manager provides the list of available devices to the context provider(SetOfAvailableDevices). The context provider sends a procedure to Embots which are in the list(UserLocalizationProcedure()). The Embots provide sensor information with executing the given procedure(UserLocalizationProcedureResult).

4) Sensor Locomotion Command and Context Information Provision: The context provider generates context information and sends it to the Sobot manager(UserLocationContext), and if needed, sends sensor locomotion commands to the task scheduler(SensorMove). Sobot manager sends the generated context information to Sobots(UserLocationContext) and task scheduler decides which Mobot should move and sends locomotion commands to the selected Mobot(SensorMoveCommand).

G. Overall structure of Ubibot

The overall structure of the ubiquitous robot system is shown Fig.4. Every component interacts with other ones through Ubi-server, which manages all components and devices in U-space and environmental information.

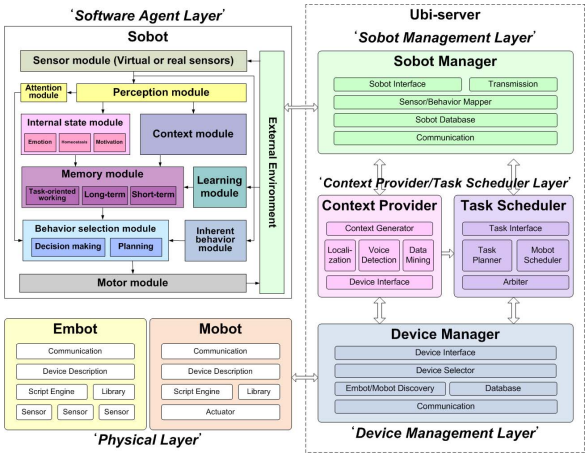


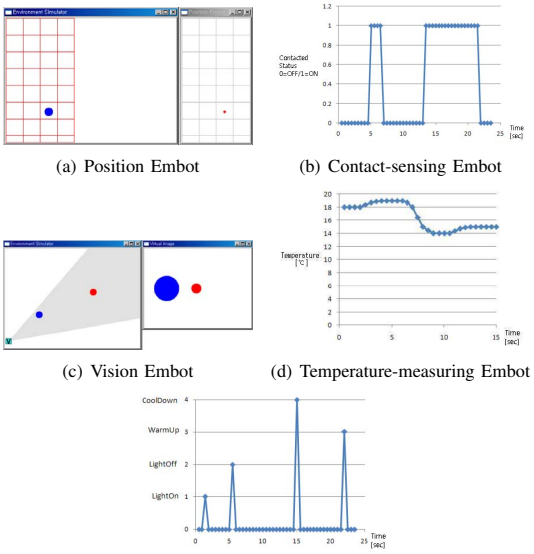
Fig. 4. Overall structure

IV. Simulations and Results

To show the feasibility of the proposed architecture, a virtual U-space with several Embots and one Mobot having a vision Embot were simulated. Three context generator components are implemented utilizing five embots, and two software agents which use the context information are simulated.

A. Virtual Embots

Five virtual Embots were simulated; position Embot, contact-sensing Embot, vision Embot, temperature-measuring Embot and sound Embot. Position Embot detects absolute position of an object, while contact-sensing Embot detects only whether it is contacted or not. Vision Embot acts like a practical camera; it has a virtual image and produces sensor information from the virtual image. Temperature-measuring Embot measures the temperature of the virtual U-space. Sound Embot recognizes the user's voice commands, and it is simulated to detect four voice commands, which are LightOn, LightOff, WarmUp and CoolDown. Simulated virtual Embots and the sensor information are shown in Fig.5.



(a) Position Embot (b) Contact-sensing Embot

(c) Vision Embot (d) Temperature-measuring Embot

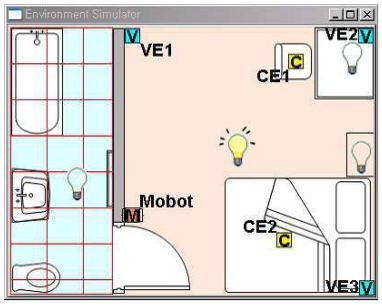
(e) Sound Embot

Fig. 5. Simulated Embots

B. Virtual U-space

Virtual U-space is set up in a rectangular room of size 8m x 6m, which is divided into two areas, a living room and a bathroom as shown in Fig.6(a). A position Embot is placed to cover the bathroom, three vision Embots(VE1, VE2, VE3) are placed in the living room and one vision Embot is attached to a Mobot. Two contact-sensing Embots(CE1, CE2) are attached to a chair and a bed in order to detect when the user sits on the chair or the bed. There are four lights, one temperature-measuring Embot and one temperature controller in the living

room. Fig.6(b) shows the symbols of devices in the virtual U-space.



(a) Environment

(b) Vision Embot, contact sensing Embot, Mobot, position Embot, Lights

Fig. 6. Virtual U-space

C. Context Generation

Three context generators are implemented for user location context, temperature context and voice command context.

User location context is generated using vision Embots, position Embots and contact-sensing Embots. Using position Embots or contact-sensing Embots, user's location is given directly from the Embot's device descriptions and sensor information, but with vision Embots, two or more vision Embots are needed. If the sensor information are not enough to obtain the user's exact location, user's location is estimated with probabilistic occupancy maps[10], and a sensor locomotion command is sent to the Mobot scheduler.

Temperature context and voice command context are generated using temperature-measuring Embots and sound Embots, respectively.

D. Sobot

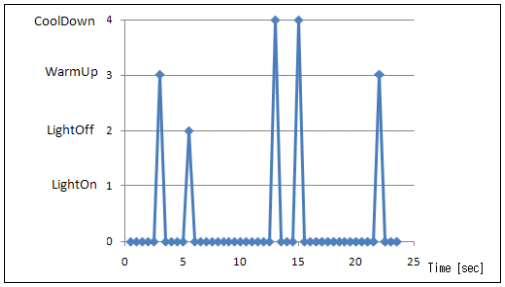
Two simple Sobots are implemented. Light-managing Sobot manages the light; turning a light on and off. Temperature-managing Sobot manages the temperature of the environment. When a user requests to turn a light on or off, the light-managing Sobot decides which light should be turned on or off utilizing user location context and voice command context. If the user goes onto bed and doesn't move for quite a while, the Sobot turns off all the lights in the room. The temperature-managing Sobot uses temperature context and voice command context to control the temperature of the environment.

E. Simulation Results

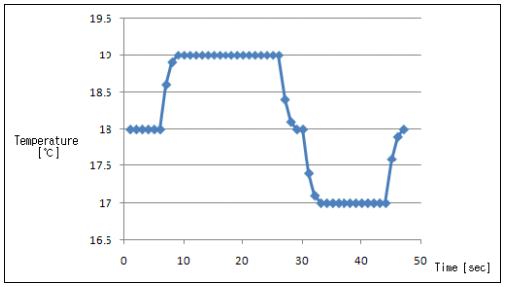
Context information were generated from the sensor data of Embots. The location of a user was estimated even when the user went out of the coverage of the Embots, and detected soon by moving Mobots to the estimated location. The context information were generated using as many sensors as possible, without interacting with Sobots.

When the user requested LightOn command, the light-managing Sobot decided which light stand should be turned on and turned on the light, which is the nearest one from the user. When WarmUp voice command or Cooldown command is recognized, the temperature-managing Sobot tried to control the temperature of the environment with the temperature controller as shown in Fig.7.

Sobots used the context information to decide services to provide, and all of these were without accessing the devices directly.



(a) Requested voice command



(b) Temperature control

Fig. 7. Simulation results of the virtual U-space

V. Conclusion

This paper proposed a multi-layered middleware architecture for ubiquitous robots, which has five layers and use context as a link between hardware components and software components for device/environment-independency and modularity.

Embots and Mobots were implemented with a script engine to execute any given procedure to have various functions and interoperability between different hardware and software platforms. Each device provides its own device descriptions. Using device descriptions, the device manager manages Embots and Mobots. The device manager provides common interfaces to all devices and makes ubiquitous robot system scalable. Generating contexts and software agents were separated to decouple software robots and hardware robots. Consequently, software robots are able to provide proper services to users when they moved to the other environments or when sensors

are added or removed. Contexts were generated using as many sensors as possible utilizing mobile robots. Mobile robots which are not performing actions created by software agents were used to gather sensor information actively.

The feasibility of the proposed architecture was shown with simulated Ubibots in a virtual U-space.

For experiments in real environments, localization modules or self-localization functionality should be provided to Embots and Mobots. The task scheduler in the simulation was implemented for only one mobile robot, but planning and scheduling for many Mobots considering their own available actions should be studied.

REFERENCES

[1] M. Weiser, "Hot topics-ubiquitous computing," *Computer*, vol. 26, no. 10, pp. 71–72, Oct 1993.

[2] ——, "Some computer science issues in ubiquitous computing," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, p. 12, 1999.

[3] J.-H. Kim, "The third generation of robotics: Ubiquitous robot," in *2nd International Conference on Autonomous Robots and Agents*, Dec 2004.

[4] ——, "Ubiquitous robot," in *Proceedings of Fuzzy Days International Conference*, Dortmund, Germany, Sep 2004, keynote Speech Paper.

[5] K. Han, Y. Kim, J. Kim, and S. Hsia, "Internet control of personal robot between kaist and uc davis," 2002. [Online]. Available: citeseer.ist.psu.edu/han02internet.html

[6] R. Luo, K. Su, S. Shen, and K. Tsai, "Networked intelligent robots through the internet: issues and opportunities," *Proceedings of the IEEE*, vol. 91, no. 3, pp. 371–382, Mar 2003.

[7] Y.-G. Ha, J.-C. Sohn, and Y.-J. Cho, "Service-oriented integration of networked robots with ubiquitous sensors and devices using the semantic web services technology," *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 3947–3952, Aug. 2005.

[8] T. O. M. Group(OMG), "The common object request broker: Architecture and specification, revision 2.3," 1999.

[9] ——, "Platform independent model(pim) and platform specific model(psm) for super distributed objects(sdo) specification, version 1.0," Nov. 2004.

[10] D. A. Isla and B. M. Blumberg, "Object persistence for synthetic creatures," in *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2002, pp. 1356–1363.