4

The Context-Awareness for the Network-based Intelligent Robot Services

Chung-Seong Hong, Kang-Woo Lee, Hyoung-Sun Kim and Hyun Kim
u-Robot Server Research Team, Electronics and Telecommunications Research Institute
Republic of Korea

1. Introduction

Recently, while the existing industrial robot market is saturated, the research about the service robot is actively progressed. In order that the service robot comes into our daily lives like the electric home appliances or mobile devices, it has to provide the intelligent services to a user.

Generally, the robot has three functional components of sensing, processing and acting. The ability of these components depends on its own hardware performance. Ubiquitous Robotic Companion (hereafter URC) is the new concept of the network-based service robot (Kim et al., 2005). In the concept of URC, a robot expands the ability of these components through the network. The robot can use not only its internal sensors but also external sensors which are embedded in the environment for sensing. A number of robots can share a high performance server to increase the processing power. Also, it can use the external actuators which are deployed in the environment, not only in its body, for acting.

These improvements can bring about the enhancement of context awareness and provide proactive services to the users. When a user requests a service, a robot is able to provide appropriate services by recognizing the user's current situations. Thus, to realize the URC, it is important to develop a context-aware system which can help robots become aware of user's situations.

The context-aware system is comprised of the various kinds of distributed computing entities. Early context-aware systems had the relatively simple structure because of being only comprised of distributed elements which communicate with the local or the remote sensors. But, as the structure of the context-aware system become complicated and various applications appear, the context-aware infrastructure has been needed in order to enhance the maintenance and the reuse of the context-aware application. It can help to remove the complexity of the application development. Therefore, the development of the context-aware framework which can provide reusable components with high-level abstraction is needed.

The aim of this chapter is to introduce the context-awareness for a network-based service robot. For this, we define the functional requirements and implementation issues of the context-aware framework and propose the layered conceptual structure. And then, we describe how proposed conceptual structure was implemented to the CAMUS (Context-Aware Middleware for URC System) which is a context-aware framework for the network-based service robots. Finally, we introduce various kinds of robot services which were developed by using the CAMUS.

2. Related works

As the context-aware computing gets more and more interests, attempts applying it to diverse kinds of research fields have been tried. However, the development of the context-aware intelligent application in the robot field has rarely been carried out. Thus, in this section we review the previous research works in the field of context-aware computing.

Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves. A system is context-aware if it uses context to provide relevant information and/or services to the user, relevancy depends on the user's task (Dey, 2000).

The context-aware system can be implemented in many ways. The approach depends on special requirements and conditions such as the location of sensors, the amount of possible users, the available resources of the used devices or the facility of a further extension of the system (Baldauf et al., 2007). A number of research groups have developed context-aware systems and applications from 1990s. However, most early researches were remained as the experimental level on developing the prototype, because various context information and its achievement technologies were considered only for specific applications at specific platforms.

Recent context-aware systems use a middleware or context-aware server architecture for separating an acquisition and use of the context. These architectures have a layered structure which has an expandability and reusability.

The architecture of the Context Managing Framework (Korpipaa et al., 2003) uses four main functional entities which comprise the context framework: the context manager, the resource servers, the context recognition services and the application. Whereas the resource servers and the context recognition services are distributed components, the context manager represents a centralized server managing a blackboard. It stores context data and provides this information to the client applications.

Another framework based on a layered architecture was built in the Hydrogen project (Hofer et al., 2002). Its context acquisition approach is specialized for mobile devices. While the availability of a centralized component is essential in the majority of existent distributed content-aware systems, the Hydrogen system tries to avoid this dependency. It distinguishes between a remote and a local context. The remote context is information another device knows about, the local context is knowledge our own device is aware of. When the devices are in physical proximity they are able to exchange these contexts in a peer-to-peer manner via WLAN, Bluetooth etc. This exchange of context information among client devices is called context sharing.

Context Toolkit (Dey et al., 2001) was developed as an intermediate dealing with context between a sensor and an application service. It collects context information from sensors, analyzes it, and delivers it to application services. The toolkits object-oriented API provides a super-class called *BaseObject* which offers generic communication abilities to ease the creation of own components.

CASS (Context-Awareness Sub-Structure) (Fahy & Clarke, 2004) proposed extensible centralized middleware approach designed for context-aware mobile applications. The middleware contains an *Interpreter*, *ContextRetriever*, *Rule Engine* and *SensorListener*. The *SensorListener* listens for updates from sensors which are located on distributed sensor nodes. Then the gathered information is stored in the database by the *Interpreter*. The

ContextRetriever is responsible for retrieving stored context data. Both of these classes may use the services of an *Interpreter*.

RSCM (Reconfigurable Context-Sensitive Middleware for Pervasive Computing) (Yau et al., 2002) separates sensors and application services, and delivers necessary context information to applications through ad-hoc network. RSCM also collects context information, analyzes, and interpret it. When this context information satisfies conditions of registered application services, RSCM delivers it to the application services. It is the client-side to make a decision whether its application services should be executed in it or not

The SOCAM (Service-oriented Context-Aware Middleware) project (Gu et al., 2004; Wang et al., 2004) introduced architecture for the building and the rapid prototyping of context-aware mobile services. It uses a central server called *context interpreter*, which gains context data through distributed *context providers* and offers it in mostly processed form to the clients. The context-aware mobile services are located on top of the architecture, thus, they make use of the different levels of context and adapt their behaviour according to the current context. Also, it proposed a formal context model based on ontology using OWL to address issues about semantic representation, context reasoning, context classification and dependency.

GAIA project (Biegel & Cahill, 2004) also developed the middleware which can obtain and infer different kinds of context information from environments. It aims at supporting the development and execution of portable applications for active spaces. In GAIA, context is represented by 4-ary predicates written in DAML+OIL. The middleware provides the functions to infer high level context information from low level sensor information, and expect the context in advance by using the historical context information.

CoBrA (Context Broker Architecture) (Chen, 2004; Chen et al., 2003) is an agent-based architecture for supporting context-aware computing in intelligent spaces. Intelligent spaces are physical spaces that are populated with intelligent systems that provide pervasive computing services to users. In the centre of CoBrA, there is an intelligent *context broker* that maintains and manages a shared contextual model on the behalf of a community of agents. These agents can be applications hosted by mobile devices that a user carries or wears (e.g., cell phones, PDAs and headphones), services that are provided by devices in a room (e.g., projector service, light controller and room temperature controller) and web services that provide a web presence for people, places and things in the physical world. The *context broker* consists of four functional main components: the *Context Knowledge Base*, *the Context Inference Engine*, the *Context Acquisition Module* and the *Privacy Management Module*.

Though many attempts for developing a context-aware system has been carried out, previous context-aware systems cannot provide a complete solution for all the essential requirements in context-aware computing. Moreover, they do not consider the robot as a computing entity which can interact with and provide services to a user. In this chapter, we describe how we developed the context-aware framework for a network-based intelligent robot.

3. Requirements and issues of context-aware framework

In this section, we define the functional requirements and implementation issues which have to be considered when we develop the context-aware framework based on the previous related researches.

3.1 The functional requirements

The functional requirements should be considered when we design the structure of the context-aware framework. The necessary functions are as follows:

1. Accessing and controlling devices
   The context-aware application can access and control various computing devices which are located in the environment of a user and robot. For this, the context-aware framework has to provide the methods of accessing and controlling these devices. With this function, context-aware application can acquire contexts and provide services with various devices.
2. Processing context
   The context-aware framework has to provide the element which has the responsibility of processing context obtained from various sensors for a context-aware application to use. The functions of this element include time stamping, interpreting, and filtering context.
3. Managing context
   The processed context has to be stored and managed with the unified context model by the context-aware framework. The managed context with the context model can be searched, accessed, and modified by the application when needed.
4. High-level context abstraction
   The context which is obtained from a sensor and managed with the context model can be processed for the high-level abstraction to provide the intelligent service by a context-aware application. This high-level abstraction should be supported by not only the context-aware application, but also context-aware framework.
5. Execution and management of context-aware application
   The context-aware framework can manage the life-cycle of the context-aware applications in order to implement the various applications easily. Moreover, the context-aware framework has to provide the functions of starting, proceeding, and termination a context-aware application.
6. Communications
   Finally, the context-aware framework has to provide the communication methods so that various context-aware elements which are located in the different physical space can be comprised of one context-aware system and they can communicate and deliver the necessary information with each other.

3.2 The implementation issues

The implementation issues are about what should be considered when we implement the context-aware framework. The selected implementation issues are as follows:

1. Support for heterogeneity
   When we develop a context-aware application using the context-aware framework, the computing entities are very various and have different resources, for example, from the resource poor sensor run on the Embedded Linux to the high performance server run on the Windows OS. The context framework has to support the heterogeneity of the OS, programming languages, and so on.
2. Support for mobility
   In the context-aware environment, a user can use not only static devices, but also mobile devices. The context-aware framework provides methods for connecting and controlling these devices dynamically.

3. Support for scalability
   The context-aware application requests and delivers the necessary information with many sensors, the actuators, and various computing entities. According to the development of multimedia, this information could be a large-scaled audio or video streaming. Therefore, the context-aware framework can resolve the scalability problem about reducing the load of the network.
4. Support for privacy protection
   Privacy is about the control of information. In a context-aware computing environment, users worry about their privacy because hidden sensors are embedded in their environment, and the information acquired by these sensors could be often shared by different applications. Users desire privacy protection because they are concerned about the possible misuse of their information. The context-aware framework provides methods for users' privacy protection.
5. Separation of concerns
   The context-aware framework provides the method for using the context which is obtained from a sensor to many context-aware applications. Thus, the acquisition of a context is separated from the use at an application for reusing context.
6. Exceptional Handling
   The context-aware framework is able to make the exception handling like the access failure or disconnection with the computing entity. Particularly, because a robot uses a wireless network in common, the problem about the disconnection and recovery has to be resolved.

4. The layered conceptual structure

The previous context-aware framework has the mutually different hierarchy structure and scope of functionality. In this section, we design the layered conceptual structure of the context-aware framework based on the defined functional requirements in section 3. The modules and operations which are located in each layer are shown in Fig. 1.

4.1 Context source and service layer

In the bottom of the layered conceptual structure, there is the *Context Source and Service Layer* which is consisted of smart devices. In this layer, there exist various kinds of sensors and actuators including robots which are special kinds of devices.
Context sources include not only hardware devices which provide explicit information given by the user or environment, but also computing sources which can be used to collect computing context from information database or web-services. Depending on the type of context, one or multiple sensors may be employed.

4.2 Device access and control layer

In the *Device Access and Control Layer*, there are the Sensor Framework Module and Service Framework Module. The Sensor Framework Module concerns how to acquire the context information from various information sources. Raw context acquisition operation captures the raw data as the value of context features. The Service Framework Module concerns how to provide services and information through actuators. Fig. 2 shows an example of context acquisition from a RFID reader.

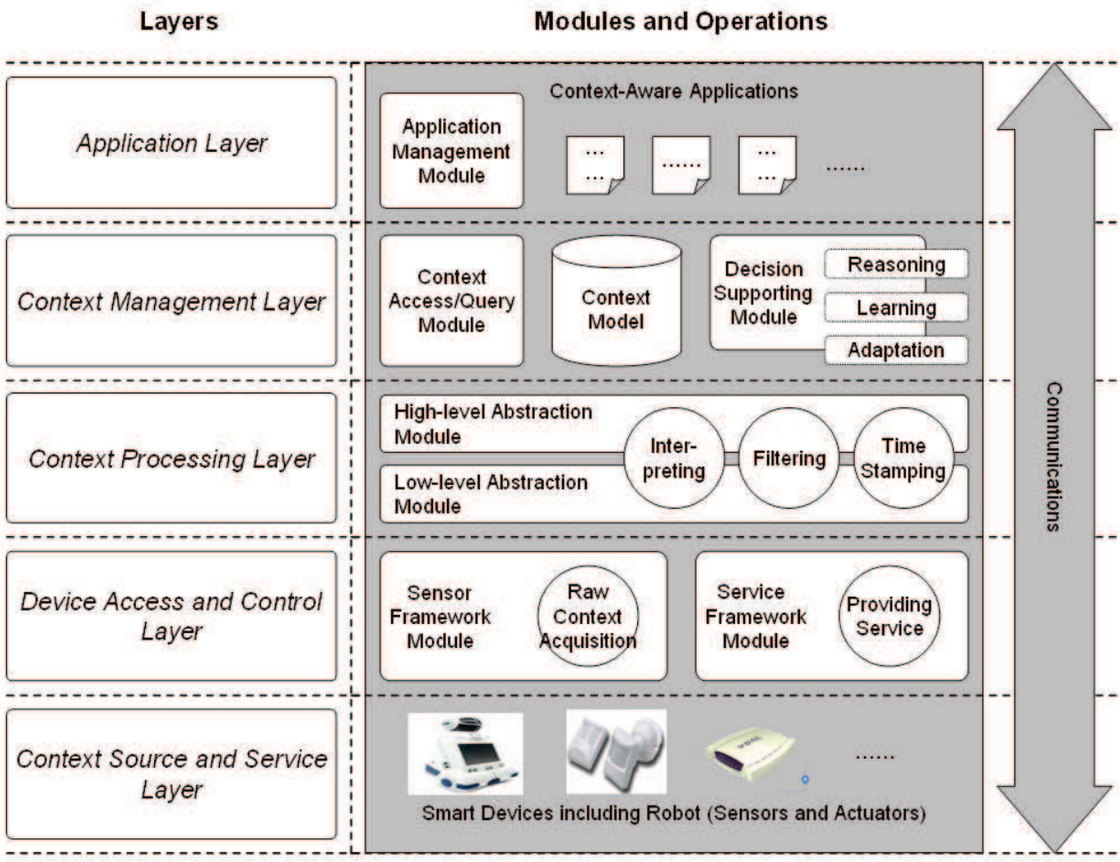

Fig. 1. The layered conceptual structure for the context-aware framework

4.3 Context processing layer

The *Context Processing Layer* concerns how to process the acquired context before storing or delivering it. The context obtained from the *Device Access and Control Layer* may include technical data which is not appropriate to use in the context-aware application directly or include unnecessary information. Usually, context processing operations are performed by the need of the context-aware application. However, the operation which is repeatedly used in many applications has to be separated from the applications and defined in the context-aware framework. After that, an application developer can use the context which is processed by the pre-defined operations without implementing them in the application. Context processing can be divided into following three operations:

1. Interpreting operation
   The Interpreting operation interprets and abstracts context so that it is available to be used in the context-aware application. For example, the current location of a user can be interpreted using the RFID tag information when a user enters the detecting range of a RFID reader which is installed in a specific location as shown in Fig. 3.
2. Time stamping operation
   Time is very important context to all applications. A context is changed while the time passes away. This is due to the fact that context information is dynamic by nature. Time

stamping operation adds time information to the context. By doing this, we can use the information of context history. Fig. 4. shows an example of time stamping operation.

3. Filtering operation
   In the contexts acquired from sensors, there can be unnecessary information. When all contexts are delivered to the components in the context-aware framework, the communication cost will rapidly increase. Moreover, it may cause the system to be down. The filtering operation reduces the amount of data by removing the context which it is not needed or duplicated. Fig. 5. shows an example of filtering operation.

The *Context Processing Layer* is comprised of the Low-level Abstraction Module and High-level Abstraction Module as shown in Fig. 1. The difference of Low-level and High-level Abstraction Modules is according to the referring to the other context which is stored using the Context Model in the *Context Management Layer*.

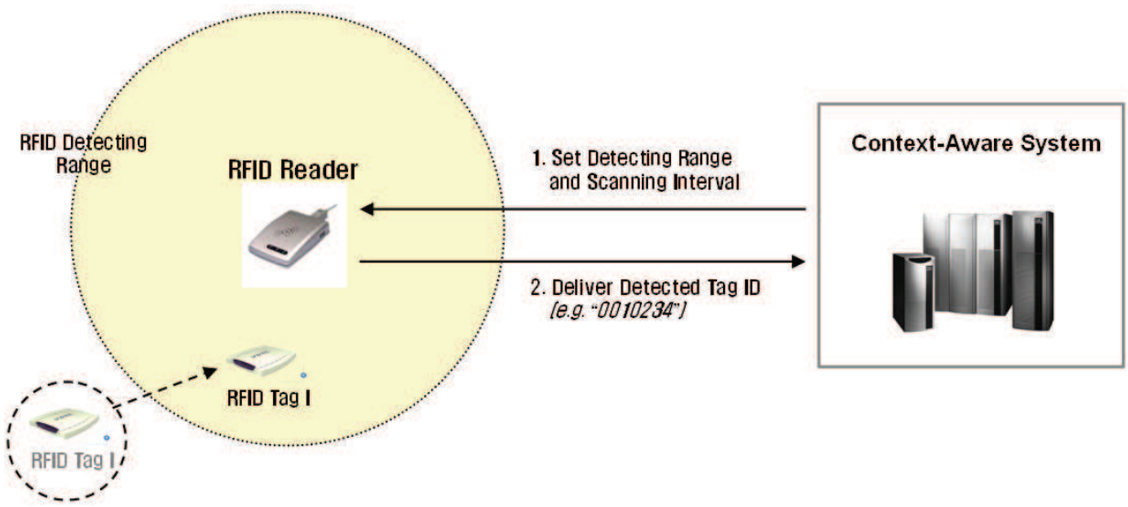

Fig. 2. An example of context acquisition from a RFID reader

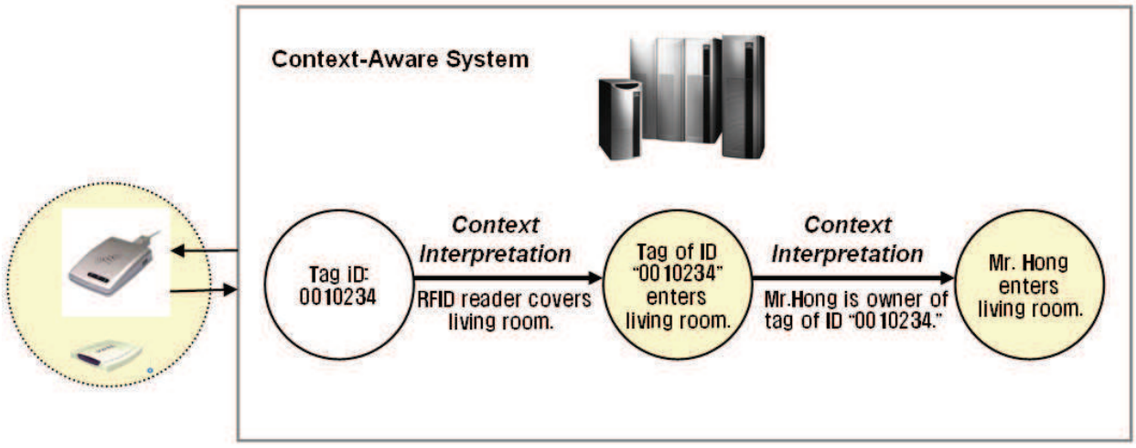

Fig. 3. An example of interpreting operation

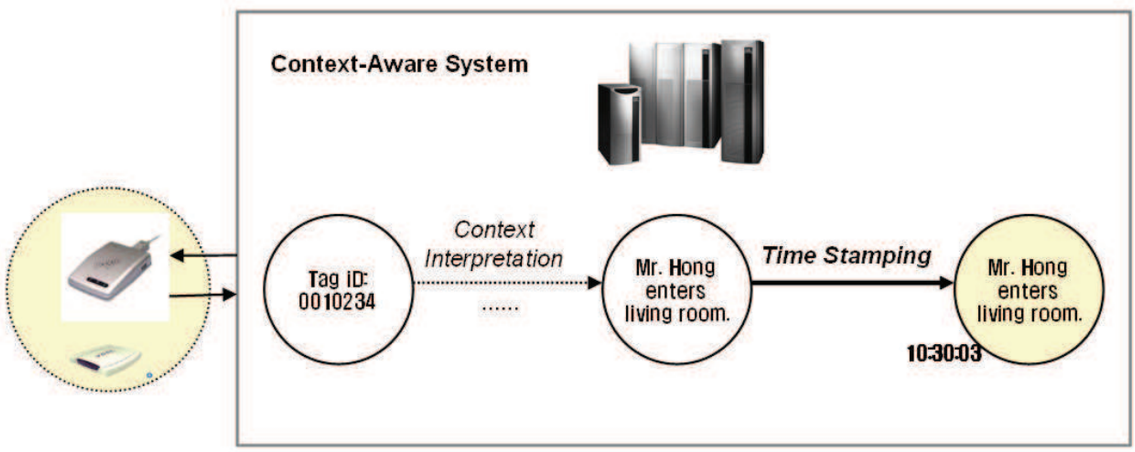

Fig. 4. An example of time stamping operation

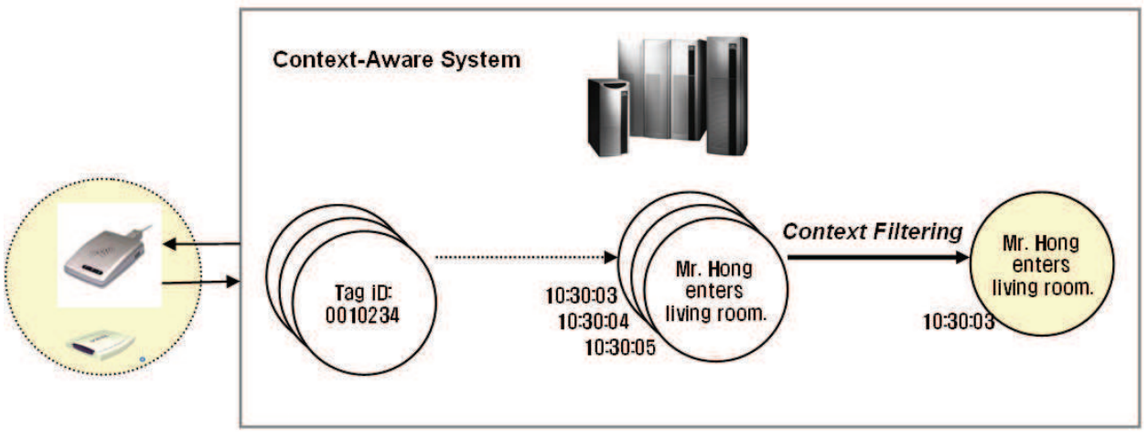

Fig. 5. An example of filtering operation

4.4 Context management layer

The *Context Management Layer* stores and manages the obtained context in order to use lately. The context is stored by using the context model. The *Context Management Layer* provides the Context Access/Query Module for accessing and modifying the context.
In addition, The *Context Management Layer* can provide the Decision Supporting Modules. The Decision Supporting Module includes the supporting tools or engine which can resolve the complex decision problems such as reasoning, dynamic adaptation, learning, and so on for general use.

4.5 Application layer

In the *Application Layer*, there are context-aware applications and the Application Management Module. The Application Management Module manages the life-cycle of an

application. The Application Management Module not only creates and finalizes an instance of application, but also manages the state of it. Moreover, the Application Management Module can provide tools for supporting event handling and rule firing. A context-aware application gets contexts from the Context Model through the Context Access/Query Interface Module or from the sensors directly through the remote-procedure call which is provided by the Sensor Framework Module.

5. The context-aware middleware for URC systems

We developed CAMUS as the context-aware server framework for a network-based robotic system (Kim et al., 2005; Hong et al, 2006). The functional architecture of CAMUS is shown in Fig. 6.

The *service agent* is a software module performed as a proxy to connect various external sensors and actuators to CAMUS. It delivers information of the sensors in environment to the CAMUS *main server*. Also, it receives control commands from CAMUS *main server* and controls devices in the environment, and conducts applications.

The *service agent* is executed in the *service agent manager*. The *service agent manger* makes accesses to adequate *service agent*s out of the currently running *service agent*s using a searching function. The *service agent manager* supports the prerequisite for the proactive context-aware services. Also, it searches, invokes, and executes necessary services for each of CAMUS *task*s. Information obtained by *service agent*s is processed by the *sensor interpreter* as the *low-level abstraction module*. This processed information will be delivered as a type of event to the CAMUS *main server* through the *event system*. Creating events indicates the changes of context in the user's or robots' situation.

The *event system* generates and manages events from physically distributed environments and is responsible for exchanging messages among CAMUS components. Above all, it delivers the events to the *context manager* and *task manage*r so that CAMUS *task*s should be able to recognize the changes of situation and further update the existing *context model*.

The *context manager* infers implicit knowledge based on the context information which is delivered through events, and it also manages this inferred knowledge. When CAMUS tasks are executed, they refer to the context knowledge managed by the *context manager*. Therefore, the *context manager* should be able to provide several main functions for utilizing the context knowledge; (a) the context modelling and the knowledge representation, (b) the management of the context data, (c) the query for the context knowledge, and (d) the inference of the implicit knowledge, and so on.

The *context model* which is represented and managed in CAMUS includes the user context, environment context, and computing device context. The user context includes user profile, user's task information, user preference, and so on. The environment context includes hierarchical location information, time, and so on. The computing device context includes information about available sensors and actuators including robots.

The *task* can be regarded as a set of work items which are required to be taken in a specific context or by the user's command. Each work item, which is a unit of action, is described by the task rules. A task rule is described by the convention of Event-Condition-Action (hereafter ECA). The action part arranges the operations of the *service agent* to be executed

when the incoming event meets conditions. The *service agent* is accessed by Task through the *service agent manager*. The ECA rule is managed by the *ECA rule engine*.
The *task manager* initiates individual tasks and manages on-going task processes. It also controls and coordinates tasks by managing the current state information of each *task*. The state of the *task* is managed by the *Finite State Machine.*
Under this functional architecture, CAMUS is mainly divided as a CAMUS *main server* and *service agent manager*s based on the embedded types in the physical environment as shown in Fig. 7. The *service agent manager* which is deployed in the local server in the home or office environment is called the *local station*. The *service agent manager* which is deployed in the robot is called the *robot station*. And, the *service agent manager* which is deployed in the mobile device is called the *sobot station*. Each type of the *service agent manager* has the different context processing ability, user interface, and program module. The CAMUS *main server* communicates with *service agent manager*s through the communication framework called *PLANET*.

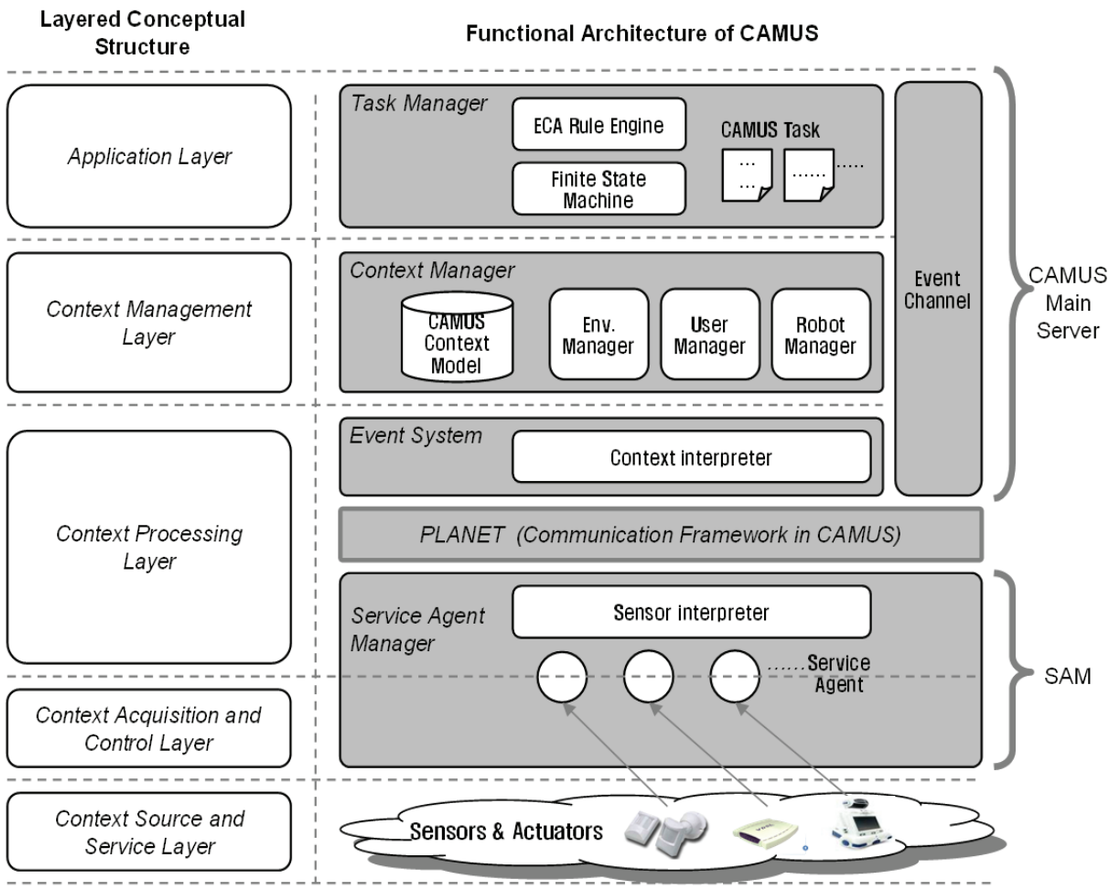

Fig. 6. The functional architecture of CAMUS

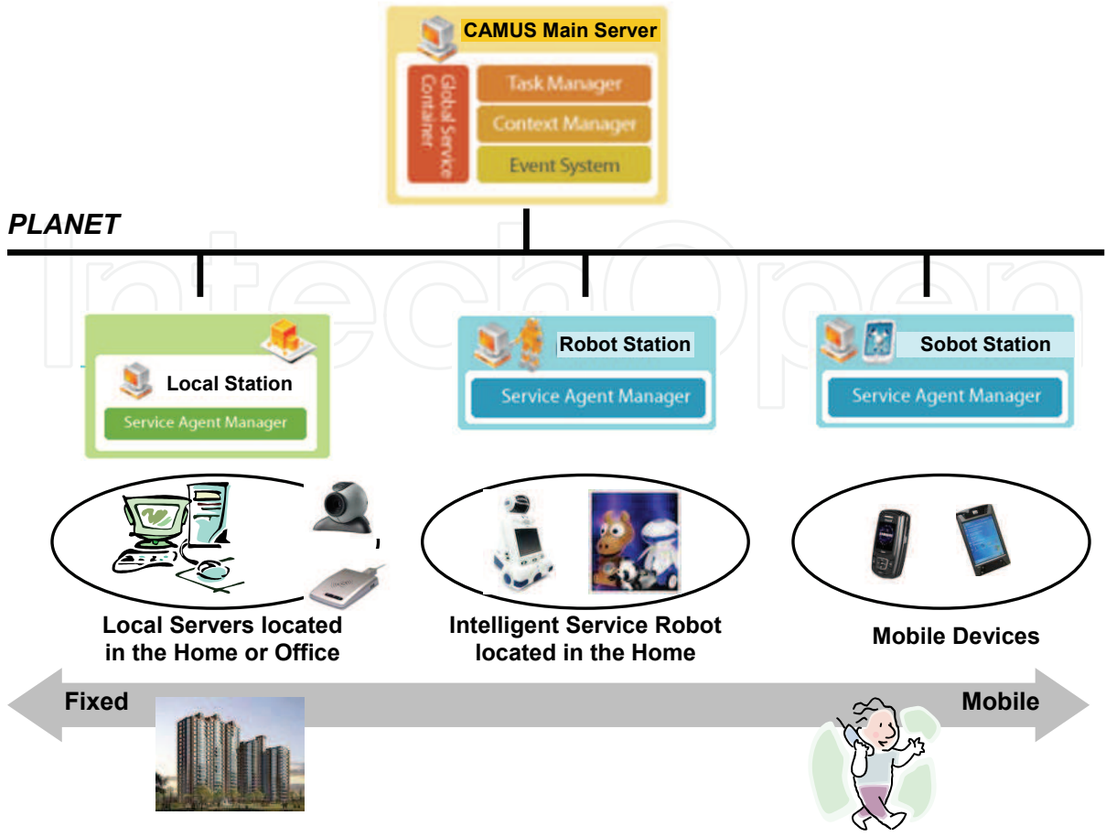

Fig. 7. System deployment in the physical environment

6. Intelligent robot services

In this section, we introduce the URC field test and several applications which are developed by using the CAMUS. Applications are the robot greeting, home monitoring, and follow-me services.

6.1 The URC field test

The first step result of developing CAMUS was validated through URC field test during two months from Oct. 2005. to Dec. 2005 (Hong et al., 2006). For URC field test, 64 households were selected from Seoul and its vicinity, where the broadband convergence network (BcN) was installed and 3 kinds of robots (Jupiter, Nettro and Roboid) were offered to them. Fig. 8 shows the URC field test structure for practical services of the CAMUS. Throughout the URC field test, we verified that the Hardware structure of a robot can be simplified by distributing robot's main internal functionalities (ex. speech recognition, and image recognition) to external servers through the network. And we also found that it is enabled to more effectively provide various kinds of services which are necessary for daily lives under the URC concept.

35 households out of 47 households' respondents showed a high degree of satisfaction, and they felt home monitoring, security, and autonomous cleaning services are most appropriate services for their daily lives. However, they felt a robot is worth only about 500 ~ 1,000

dollars with its performance, which implies that the robot market targeting households needs to make an effort to stabilize the robot price reasonably.

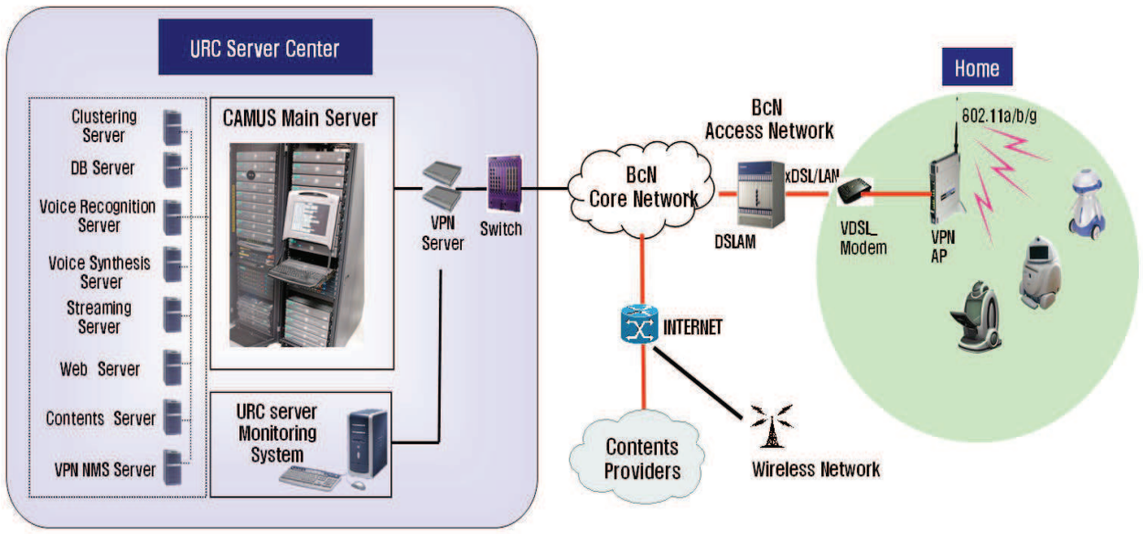

Fig. 8. System structure of the URC field test (Hong et al. 2006)

6.2 The robot greeting service

This is the robot service which was developed for the URC field test (Hong et al., 2006). The used robot called Jupiter was made by the YUJIN ROBOT, Co. and the application was developed by using the CAMUS.

In the scenario of the robot greeting service, a user arrives at home and goes to the living room. The RFID reader deployed in the living room recognizes the user's entrance from outside of the home. The RFID reader delivers the entrance information through the event in the CAMUS. CAMUS *task* orders the robot to move to the living room. The robot delivers an event about navigation completed to the CAMUS *task* through the event channel after moving. The robot says the greeting message using the Text-To-Speech service. The robot shows a message which is left from the user's family. The CAMUS task orders the robot to move to the standby-position and wait for user's command. The snapshots of this service are shown in Fig. 9.

Many kinds of sensors and actuators were used for this service. As a sensor, the RFID reader was used to recognize the location of a user and the microphone which was embedded in the robot were used for the Automatic-Speech-Recognition service. The speaker used by the Text-To-Speech service, robot navigation service, and the LCD monitor were used as the actuators.

6.3 The home monitoring service with the mobile phone

We think that the home monitoring service is one of killer applications in the service robot field. The robot navigation service and video streaming technology enable this service.

In the scenario of the home monitoring service, a robot watches the inside of home. When a thief makes an appearance during the robot's home monitoring service, the robot detects a movement of the thief and records it. A warning message is sent to a user from the robot,

and a user can watches the recorded video streaming by using the mobile phone. The snapshots of the home monitoring service are shown in Fig. 10.

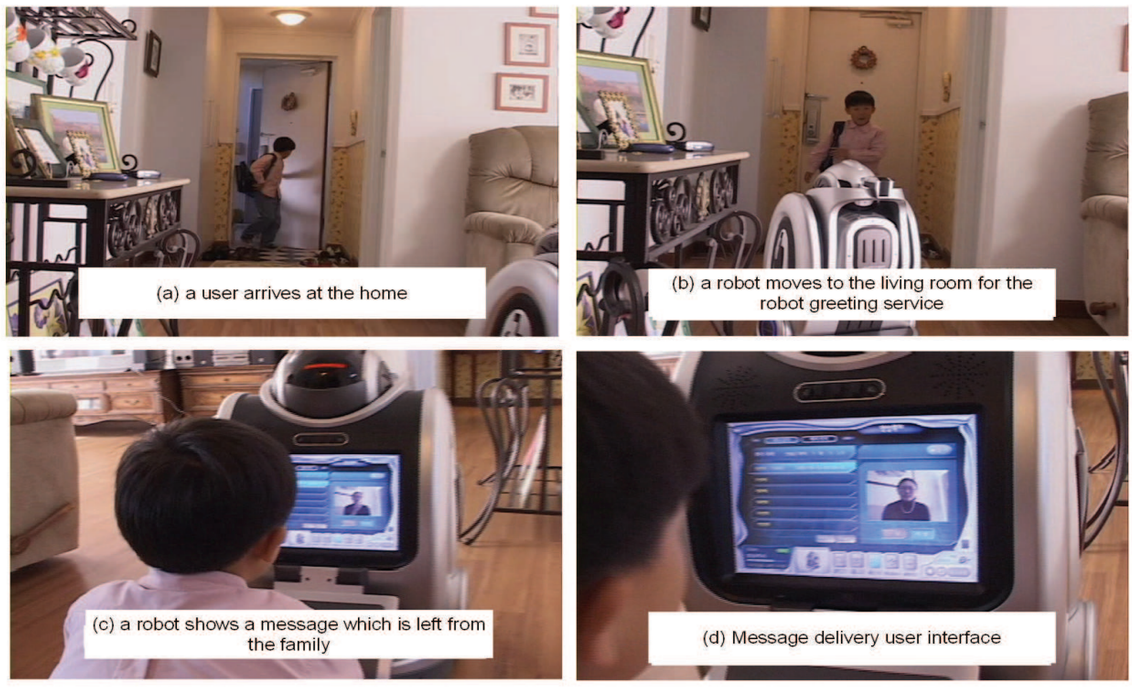

Fig. 9. Snapshots of the robot greeting service (Hong et al., 2006)

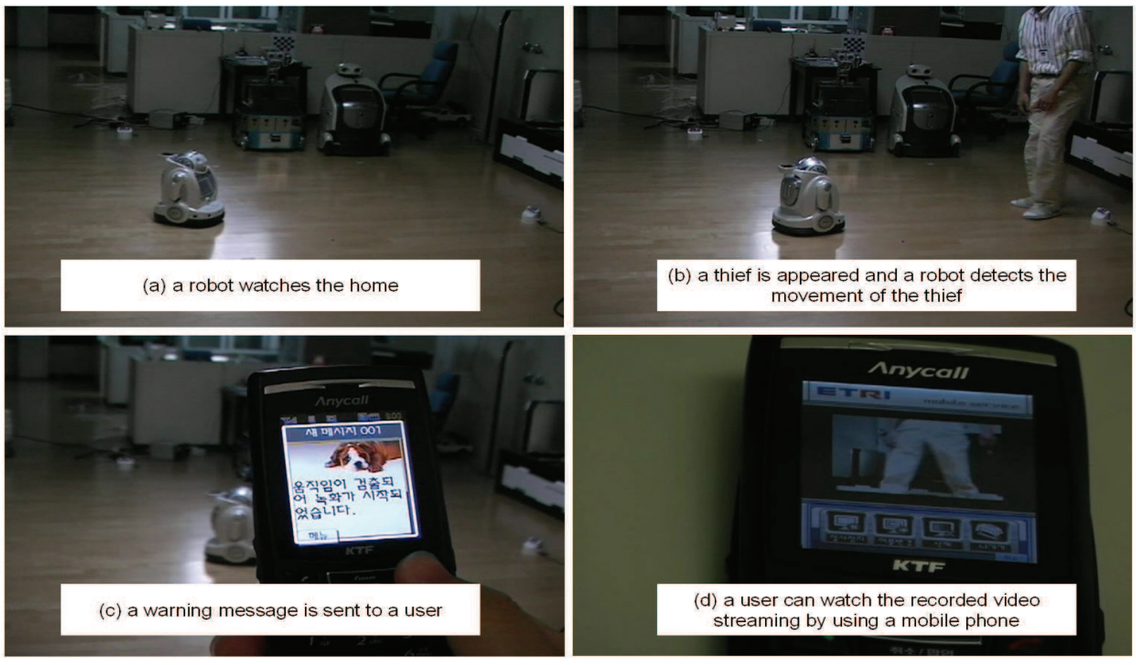

Fig. 10. Snapshots of the home monitoring service

6.4 The follow-me service

The follow-me service is a traditional ubiquitous computing application. By using the CAMUS, we can easily implement the follow-me service. We enhanced the follow-me service with the concept of the software robot (hereafter, sobot). Sobot is a virtual robot which is located in the virtual place. Sobot is responsible for the interaction between the CAMUS and users when they can not be provided services from a real robot. The snapshots of the follow-me service shown in Fig. 11.

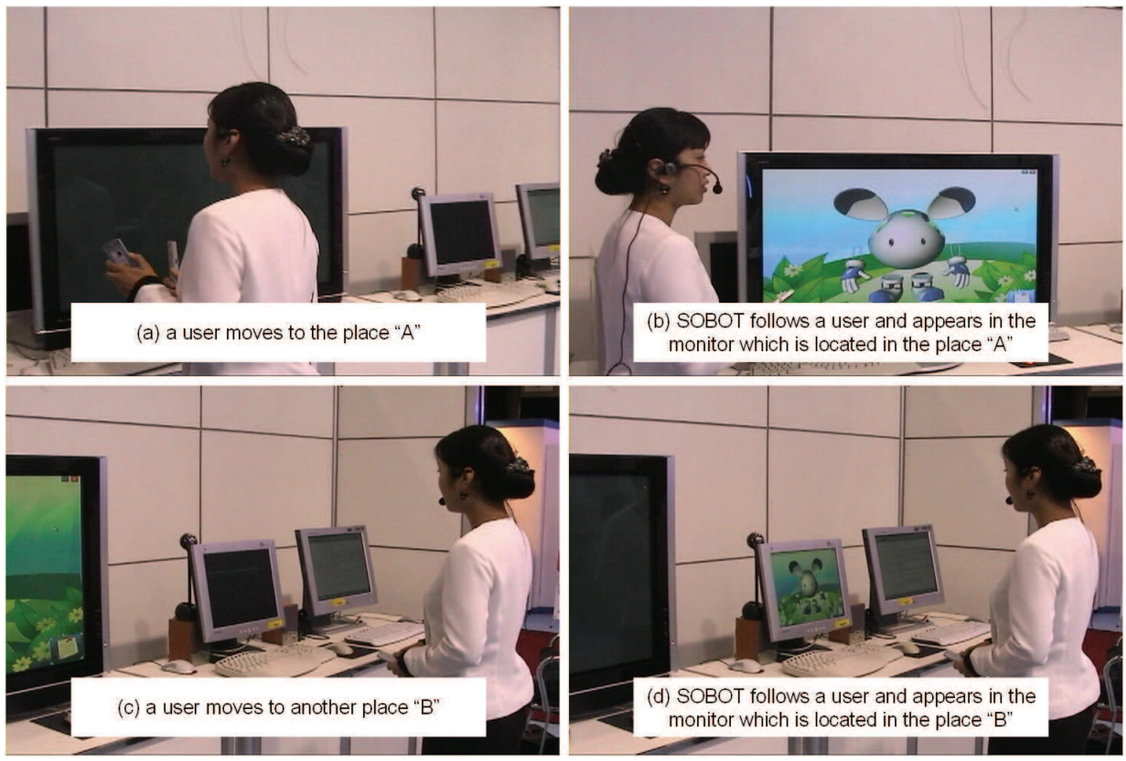

Fig. 11. Snapshots of the follow-me service

7. Conclusions

As the context-aware computing gets more and more interests, attempts applying it to diverse kinds of research fields have been tried. However, the development of the context-aware intelligent application in the robot field has rarely been carried out. In order that the service robot comes into our daily lives like the electric home appliances or mobile devices, it has to provide the context-aware intelligent services to a user.

The aim of this chapter is to introduce the context-awareness for a network-based service robot. For this, we define the functional requirements and implementation issues of the context-aware framework and propose the layered conceptual architecture. And then, we describe how proposed conceptual architecture was implemented to the CAMUS which is a context-aware framework for the network-based service robots. Finally, we introduce various kinds of robot services which were developed by using the CAMUS. As a result of

this work, we can easily develop various kinds of context-aware intelligent services by using the CAMUS.

There are many works that have to be researched. One of the remained works is the navigation of a robot. The technology of the robot navigation is not enough to be applied to a commercial product. Also, we have to consider the implementation issues mentioned in section 3 and should answer how to resolve it. I believe, nevertheless, that the network-based intelligent service robot will be very promising research field and it can improve the life of the human in the future.

8. References

Baldauf, M., Dustdar, S. & Rosenberg, F. (2007). A Survey on Context-Aware Systems, *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 2, No. 4, pp. 263-277, ISSN 1743-8225

Biegel, G. & Cahill, V. (2004). A Framework for Developing Mobile, Context-aware Applications, *IEEE International Conference on Pervasive Computing and Communications*

Chen, H. (2004). An Intelligent Broker Architecture for Pervasive Context-Aware Systems, *PhD thesis*, University of Maryland, Baltimore County

Chen, H., Finin, T. & Joshi, A. (2003). An ontology for context-aware pervasive computing environments, The Knowledge Engineering Review, Vol. 18. No.3

Dey, A.K. (2000). Providing Architectural Support for Building Context-Aware Applications, *PhD thesis*, Georgia Institute of Technology

Dey, A.K., Abowd, G.D. & Salber, D. (2001). A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications, *Anchor article of a special issue on Context-Aware Computing, Human-Computer Interaction (HCI) Journal*, Vol. 16

Fahy, P. & Clarke, S. (2004). CASS–a Middleware for Mobile Context-Aware Applications. In Workshop on Context Awareness, *MobiSys*

Gu, T., Pung, H. K. & Zhang, D. Q. (2004). A middleware for building context-aware mobile services, *In Proceedings of IEEE Vehicular Technology Conference (VTC)*, Milan, Italy

Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G. & Altmann, J. (2002), Context-Awareness on Mobile Devices – the Hydrogen Approach, *In Proceedings of the 36th Annual Hawaii International Conference on System Sciences*

Hong, C-S., Cho, J., Lee, K-W., Suh, Y-H., Kim, H. & Lee, H-C. (2006). Developing Context-Aware System for Providing Intelligent Robot Services, *Proceedings of European Conference on Smart Sensing and Context (EUROSSC2006)*, LNCS 4272, pp. 174–189, ISBN 3-540-47842-6, Enschede, Netherlands, October 2006, Springer

Kim, H., Cho., Y.J. & Oh, S.R. (2005). CAMUS - A middleware supporting context-aware services for network-based robots, *IEEE Workshop on Advanced Robotics and Its Social Impacts*, Nagoya, Japan

Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H. & Malm, E.-J. (2003). Managing Context Information in Mobile Devices, *IEEE Pervasive Computing*, July, pp. 42-51

Wang, X.H., Zhang, D.Q., Gu, T. & Pung, H.K. (2004). Ontology based Context Modeling and Reasoning using OWL, *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*

Yau, S.S., Karim, F., Wang, Y., Wang, B. & Gupta, S. (2002). Reconfigurable Context-Sensitive Middleware for Pervasive Computing, *IEEE Pervasive Computing joint special issue with IEEE Personal Communications*, Vol. 1, No. 3

Advances in Service Robotics

Edited by Ho Seok Ahn

ISBN 978-953-7619-02-2
Hard cover, 342 pages
Publisher InTech
Published online 01, July, 2008
Published in print edition July, 2008

This book consists of 18 chapters about current research results of service robots. Topics covered include various kinds of service robots, development environments, architectures of service robots, Human-Robot Interaction, networks of service robots and basic researches such as SLAM, sensor network, etc. This book has some examples of the research activities on Service Robotics going on around the globe, but many chapters in this book concern advanced research on this area and cover interesting topics. Therefore I hope that all who read this book will find lots of helpful information and be interested in Service Robotics. I am really appreciative of all authors who have invested a great deal of time to write such interesting and high quality chapters.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen