Enhancing the Sentience of URC using Atlas Service-Oriented Architecture

Hyun Kim, Young-Jo Cho
Intelligent Robot Research Division
ETRI, Daejeon, Korea
{hyunkim, youngjo}@etri.re.kr

Sumi Helal, Hen-I Yang and Raja Bose
Department of Computer & Information Science & Engineering,
University of Florida, Gainesville, FL 32611, USA
{helal, hyang, rbose}@cise.ufl.edu

Abstract— **In a conventional robotic system, all sensing, actuation and computer processing are done on the robot. Such stand-alone, all inclusive intelligence design limits the possibilities as far as functionality and applications, and at the same time increases the cost of the robot. Recently, pervasive computing technologies have opened new possibilities for lending robots cooperative active spaces, in which sensors, actuators and smart devices can collaborate with the robot as a mobile sensing platform, a complex and sophisticated actuator and a human interface. URC (Ubiquitous Robotic Companion) is a network-based robotic system in which different kinds of robots are connected to the pervasive computing environment. This paper proposes a system architecture to integrate URC robots with pervasive computing environments using University of Florida Atlas service-oriented middleware architecture. We discuss the architecture of the overall Atlas based URC system and describe its implementation. We show how the integrated URC system is enabled to provide better services which enhance the sentience of URC and improve the physical interaction between the smart space and the robot, on the one hand and users, on the other.**

I. INTRODUCTION

A URC (Ubiquitous Robotic Companion) is a new concept for a network-based robotic system. The basic approach of URC is to distribute the robot's three main functions – sensing, processing and action - through the network (Fig. 1). URC robots may have minimum sensing and processing capabilities. They utilize external sensors in the URC environment rather than expanding their internal sensors. They also use the URC server, which is connected via a broadband network, for overcoming their on-board memory and processor constraints[1]. ETRI has developed URC core technologies and applied them to URC services in real fields since 2004. Though we verified URC concept throughout field tests, we cannot say that the URC services were sufficient to meet user expectations. One of many reasons is that we could not fully utilize external sensors and actuators in the environment, even though it is an important aspect in URC. If the sensing functions could be enhanced by using external sensors, the sentience capability of the URC robot would be dramatically improved. In other words, if we can integrate URC robots with a smart space, it is obvious that we can provide much better URC services. However, it is not easy to build a smart space for URC, because robot environments are diverse, and subject to dynamic change, even while the robot is running.

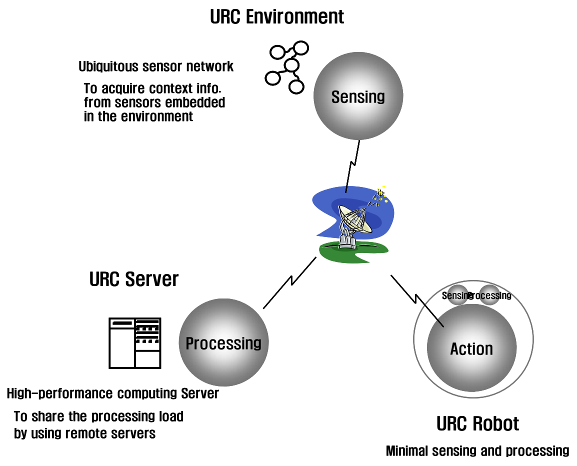
URC Environment
Ubiquitous sensor network
To acquire context info. from sensors embedded in the environment
Sensing
URC Server
Processing
High-performance computing Server
To share the processing load by using remote servers
Sensing
Processing
Action
URC Robot
Minimal sensing and processing

Fig. 1 The concept of the URC

The University of Florida Mobile and Pervasive Computing Laboratory has accumulated significant experience in designing and building smart spaces throughout the years. In particular, the technology developed has culminated in the grand opening of the Gator Tech Smart House (GTSH) project, a 2500 sq. ft. stand-alone intelligent house designed for assistive independent leaving for elderly residents. The primary technical foundation of GTSH is the Atlas Platform [2],which includes plug-and-play Atlas sensor and actuator nodes, Atlas middleware that employs service oriented architecture, and a set of intuitive tools for easy configurations and fast implementation of smart spaces.

The collaboration between ETRI and UF brings together the best technologies in robotics and pervasive computing to create extremely powerful synergy. The result is a fast and easily deployable smart space that robots can collaborate with and surrogate to, which can offer and extend its sensing and processing capabilities. On the other hand, the introduction of URC into smart houses brings unforeseen capabilities to perform complex actuations on mobile platforms that can provide better assistance to the elderly that require mobility and human touches. In this paper, we discuss on the system architecture to integrate URC robots with the smart space in order to enhance the sentience of URC.

The remainder of the paper is organized as follows. Section 2 reviews related works. Section 3 describes the conceptual architecture of the proposed system. Section 4 introduces an Atlas node which is a bridge between the physical and digital environments. Section 5 describes Atlas service-oriented middleware. Section 6 covers the URC Server which is integrated with Atlas middleware. Finally, Section 7 concludes this paper with some remarks.

II. RELATED WORK

Most previous approaches to the service robotics have focused on giving standalone robots autonomy. However, most of current service robots are used by borrowing human teleoperator's intelligence or in a simple environment in which only low-level intelligence is needed to perform actions. The reason is that the full autonomy in a standalone robot is too far from being achieved with current technology. There have been some researches to try to utilize the environment together to overcome these limitations. However, these architectures are based on a premise that the environment passively supports robot's physical tasks such as navigation and manipulation.

The pervasive computing technologies have opened new possibilities for providing robots with active spaces, in which sensors, actuators and smart devices can collaborate with robots. Examples of pervasive computing technologies have been deployed to create intelligent environments include homes [3, 4, 5], offices [6], classrooms [7] and hospitals [8]. Depending on the environment that they have been deployed to and the primary goals of the system, pervasive computing technologies provide services in location tracking, context gathering and interpretation, human gesture recognition, activity planning and many more. These services have been created to gather data in intelligent environments, process and make decisions based on the collected data and information, and then direct the actions of the devices and actuators in the same environments. However, much of the sensed data and information is also helpful in assisting robots.

There are existing projects that attempt to enhance robots using pervasive computing technologies. A mobile kitchen robot uses Player/Stage/Gazebo software library to create a middleware and supports integration into ubiquitous sensing infrastructures to enhance the interfacing, manipulation and cognitive processing of the robot[9]. The networked robot project of the Japanese Network Robot Forum [10] has been carried out since 2004. It considers sensors and actuators in the environment as unconscious robots which collaborate with software robots and physical robots. The Korean URC project also is one of major efforts to improve robots' intelligence using pervasive computing technologies. It enables robots to reduce costs and improve service qualities by taking full advantage of ubiquitous network environment and the networked server framework. This paper proposed the system architecture for integrating robotic and pervasive computing technologies, as a part of the URC efforts.

III. SYSTEM OVERVIEW

We consider the following requirements for the architecture of the proposed system:

- Separation of concern: Sensors and actuators in the smart space are separated from robot applications. That is none of the sensors and actuators or the robot is essential to the other's operation.
- Easy to construct: it should be easy to deploy, set up and maintain a smart space with minimal efforts.
- Ease of integration: it should be simple to integrate existing devices into the environment.
- Modularity: The devices in a smart space should be modular and also reconfigurable during runtime.
- Scalability: The scalability is also considered to allow integration of local spaces into large systems.
- Compatibility: The architecture should have compatibility with previous URC's as possible.

Figure 2 shows the conceptual architecture of the proposed system. It consists of four-layers: physical layer, physical interface layer, middleware layer and application lever. The layered structure allows developers to focus their attention on one layer without having to worry about the details of other layers.

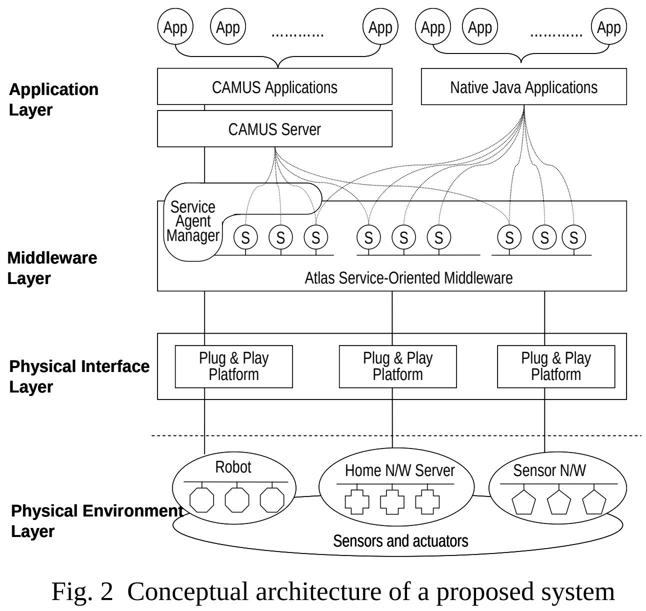

Fig. 2 Conceptual architecture of a proposed system

The *physical layer* consists of different kinds of devices and systems such as robots, which have sensing and actuating capabilities to gather information and control devices.

The *physical interface layer* bridges physical and digital spaces. It brings devices and systems deployed in the physical space into the URC by representing them as software services. This enables applications to discover and use existing services and their associated knowledge. It is also responsible for relaying information from the devices to the middleware, and controlling their operations using instructions issued by the middleware layer. Simply powering on a device makes it automatically available as a software service in the middleware layer. This plug-and-play capability is at the core of what makes it easy to construct and maintain the digital space.

The *middleware layer*, based on the service-oriented architecture (SOA), manages the lifecycle of the software services. It provides the service discovery, composition, and invocation mechanisms for applications to locate and make use of particular sensors or actuators. This enables developers in the application layer to discover the availability of specific services in the space and efficiently compose these services into applications. All services export a set of interfaces exposing their functions to applications. Various implementations may support the same interface, but as long as the interface remains unchanged, the differences in implementation should not affect applications.

In *application layer*, developers have the option of creating applications either by directly accessing the device services through native SOA mechanisms or by using a robot application framework which is called as CAMUS (Context-Aware Middleware for URC Systems)[1]. It collects contextual information from robots and the environment, and triggers appropriate events in relevant applications. It also provides the runtime environment for server-based robot applications.

IV. Atlas Node As Physical Interface

Atlas nodes, which are located in the physical interface layer, provide a bridge between the physical and digital worlds, allowing seamless integration of sensors, actuators and smart devices into pervasive computing systems.

The Plug-and-Play capability of Atlas nodes significantly simplifies and speeds up the process of creating smart spaces. The firmware running on each node is responsible for seeking out the centralized server hosting the middleware. When a node powers up, it locates the server and exchanges configuration data. Based on the configuration data, the middleware fetches service bundles associated with each connected device. Once device bundles are loaded successfully, the node is ready to relay sensor readings and accept actuator commands.

Atlas node hardware uses a stackable, swappable architecture to support on-the-fly reconfiguration without requiring any software changes from the user. Nodes are comprised of three layers: the Core Processing Layer that operates the node, a Device Interface Layer that allows various sensors and actuators to be connected, and a Communication Layer for connecting with the higher levels of the middleware. Fig. 3 shows the configuration of a typical Atlas node.

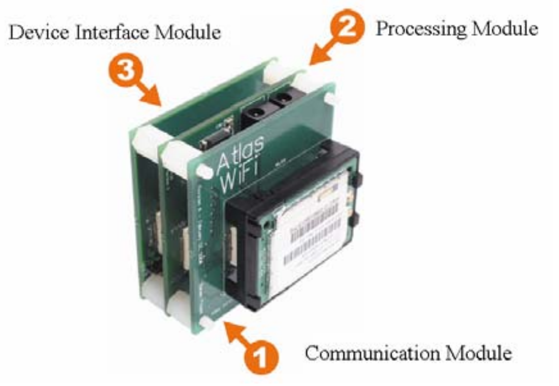

Fig. 3 Atlas Node Configuration

V. Atlas Service-Oriented Middleware

The Atlas Middleware is built on top of open service gateway initiative (OSGi) and connects to Plug-and-Play Atlas nodes to represent both dumb objects and intelligent devices as services. Each of them can further utilize the support from other existing services. The well-defined service interfaces provide a uniform and homogenous way to access large number of heterogeneous entities.

The use of SOA allows programmers to take advantage of mechanisms such as web services and related work in service composition, reconfiguration and protocol verifications. This provides open-ended support for continuous operation over entity failure or dynamic joining and departing. The Atlas middleware includes the following modules:

- Configuration and Administration Tool.
- Communication Module
- Data Processing Module
- Safety Module

A. *Configuration and Administration Tool*

The suite of configuration and administration tool, including the network manager, network viewer, configuration manager and bundle repository, provides a convenient mean for configuring Atlas nodes and managing their network connectivity. Once properly connected and configured, Altas nodes are able to automatically translate their connected hardware devices into software services.

The Network Manager is responsible for keeping track of all the nodes in the network, both existing and newly joined, and the device services they offer. When a node joins the network, it handshakes with the Network Manager, then uploads its configuration. At this point, control is passed to the Configuration Manager, which registers the device services and activates the node. The devices are then ready to be used by applications and other services.

The Network Viewer provides a web-based front-end to the Network Manager, which visualizes the current network status. It displays the list of active nodes and a short summary of their connection statistics and histories. By clicking on a particular node, users are able to view details such as configuration parameters, the properties of sensors or actuators connected and the services offered.

The Configuration Manager encapsulates all the methods for recording and manipulating node settings. When a new node uploads its configuration file, the Network Manager passes it on to the Configuration Manager, which then parses the file and accesses the Bundle Repository to get the references for service bundles required by the connected devices. It then loads these services bundles into the OSGi framework, thereby registering the different device services associated with the node.

The Bundle Repository manages the various device service bundles required by physical sensors and actuators deployed in the sensor network. The Bundle Repository eliminates the need for each node to locally store bundles. Instead, the Configuration Manager retrieves references to the bundles from the Bundle Repository when new nodes join. It enables nodes with limited storage to support a larger number and greater variety of devices, and also grant nodes the access to the most up-to-date version of service bundles.

A web-based front-end allow users to view and modify lists of the available service bundles, the physical devices they represent and other details such as the version numbers and dates uploaded. Users are able to add, delete and update service bundles, and synchronize with other repositories.

B. *Communication Modules.*

Instead of implementing a singular communication manager that oversees all the communications within and connected to our Atlas middleware, an array of communication modules for effective communications in a heterogeneous environment is used. With the number and diversity of the entities in a pervasive environment, it is unrealistic to expect one single communication protocol to work on vast number of diverse entities from different vendors. For internal communications, services can use the OSGi wiring API for inter-bundle communications. For external communications, Atlas middleware currently support three modules realizing three different protocols, the telnet/ssh client, HTTP client and web service interfacing module running SOAP over HTTP. These modules allows the services and entities reside in the server to communicate with non-Atlas based sensors and actuators, as well as external services and systems such as existing business applications.

C. *Data Processing Modules.*

Simply retrieving data from each connected sensor and device is neither scalable nor reliable in a normal setting for pervasive computing. Two data processing modules are implemented to address these issues: the On-board Processing module installs a digital filter on Atlas nodes to aggregate data, reduce noise, and allows applications to delegate some decision-making to the nodes; and the Virtual Sensors module allows continued operation amid device failures by providing a compensation mechanism and data quality indicator.

D. *Safety Module*

The highly personalized and failure-prone nature of pervasive computing services make safety features a crucial part of their design. The Space Safety Monitor uses the Context Manager to proactively monitor currently active contexts and ensure that, should any dangerous impermissible contexts become active, they will be properly handled. The Device Safety Enforcer ensures that each device only operates within a safe range and with non-conflicting operations. The Service State Manager ensures that services honor the desires of users, and can be safely terminated should the need arises.

VI. URC Application Framework

Atlas service-oriented middleware is integrated to the URC application framework known as CAMUS (Context-Aware Middleware for URC Systems). The CAMUS connects and controls many different kinds of robots. It acquires sensing information from robots and environments, plans tasks based on the current context and finally sends command messages to robots. It also has some functions including voice recognition, voice synthesis and image recognition, which are heretofore executed in a robot itself.

CAMUS consists of three main modules: Service Agent Managers, CAMUS server and Planet. The Service Agent Manager (SAM) is the robot-side framework to send robot's sensing data to the CAMUS server and receive control commands from the CAMUS server. The CAMUS server is the server-side framework to manage the information delivered from SAMs, generate and disseminate appropriate events according to the context changes, and finally execute server-based robot tasks. The Planet is the communication framework between SAMs and the CAMUS server.

A. *SAM: Service Agent Manager*

The *service agent* is a software module performed as a proxy to connect various sensors and devices to CAMUS server. It raises software events to notify CAMUS server that its device detects some noticeable changes and receives requests from CAMUS server to actuate its device. SAM is a software container of these service agents that reside in a robot or a space, and manages them. SAM plays a central role in the integration of Atlas and CAMUS systems. The collaboration between smart spaces and robots are facilitated by the communication between Atlas middleware and SAM. Fig. 4 shows the architecture of SAM.

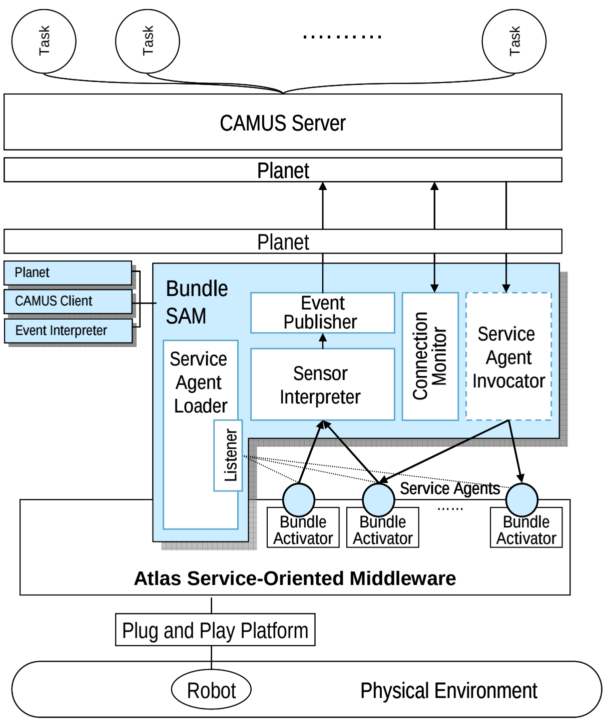

Fig. 4. Architecture of SAM

When a new device is deployed in the space, the plug-and-play capability ensures that a service bundle is activated in the Atlas middleware; at the same time, the *Service Agent Loader* detects this event from Atlas Middleware and it registers this service to the CAMUS server so that CAMUS tasks use this service. Similarly, when an existing device is removed from the space, the corresponding service bundle automatically expires; while the service agent loader also detects this event and un-registers them from the CAMUS server.

The integration between the systems requires service bundles to implement interfaces to both communicate with other services in the Atlas middleware, as well as SAM. The bundles run in the OSGi framework, but also have the capabilities of regular service agents.

The *Service Agent Invocator* allows the tasks at CAMUS server to invoke methods at service agents. A key feature of the Service Agent Invocator is supporting the asynchronous method invocation. It takes parts in necessary scheduling and thread managements in handling asynchronous invocation.

The *Connection Monitor* plays a key role in handling the disconnection. It continuously monitors the connection to the CAMUS server and takes appropriate actions for the reconnection whenever it detects a disconnection.

Any events raised by a service agent run through the *Sensor Interpreter* in SAM. The sensor interpreter examines each event, passing, dropping, refining, or aggregating them. Sometimes the event interceptor inserts new events based on the event it examines. By this way, any duplicated or unnecessary events are filtered out at SAM, reducing network and computational overhead at CAMUS server. The event that survives the run-through is sent to the Event Publisher. The event publisher delivers it to the corresponding event queue at CAMUS server. Any subscriber to the queue will be received it.

B. CAMUS Server

The CAMUS server is a framework which collects and manages information delivered by SAMs, generates and disseminates appropriate events to tasks according to the context change. It also provides the task engine to execute server-based robot applications. Fig. 5 shows the architecture of the CAMUS server.

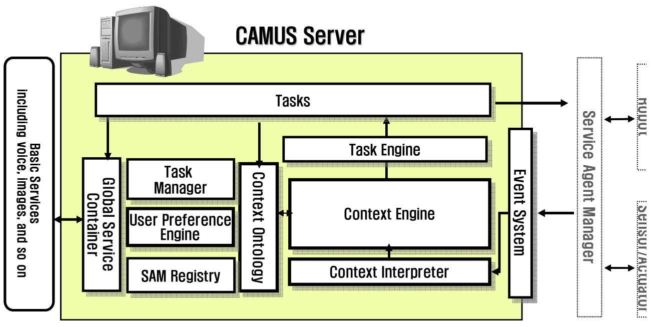

Fig. 5. Architecture of CAMUS server

Information from SAMs is delivered and managed throughout the *Event System* in the CAMUS server. It is analyzed into high-level context information by the C*ontext Engine*. The CAMUS supports the OWL-based ontology model and provides application developers with common context ontology. That is, application developers can make a domain ontology model by using the CAMUS common ontology’s semantic context entities such as place, user, device and activity. These concepts are focal and adequate to semantically represent any context information in pervasive environments[11]. Although the various reasoning engines can be plugged in the CAMUS server, we now use the JENA [12] for ontology reasoning. The application context in CAMUS tasks can be referred or changed through JENA API and RDQL.

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 *Task Engine* executes each of tasks, which are executed based on the state transitions and the ECA (Event-Condition-Action) rules. Whenever this ECA rule is fired, relevant context knowledge is referred through the context manager.

User preference engine manages user preferences according to user's behavioral histories.

Finally, the *Basic Service Container* provides basic functions of robots such as voice recognition, image recognition, motion control, among others.

Fig. 6 shows URC robots connected to the CAMUS server for practical services.

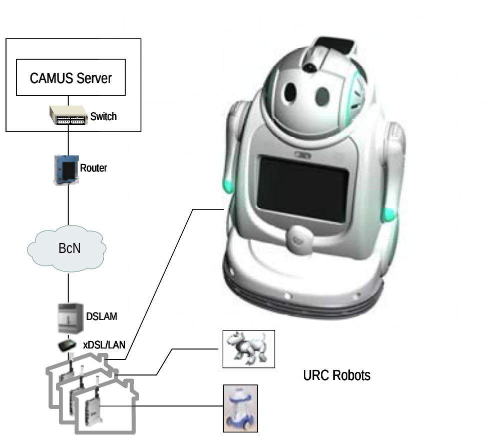

Fig. 6 URC robots connected to CAMUS server

VII. Conclusions

The integration between robots and smart spaces makes robots more intelligent, and also makes a smart space more interactive. However, the major difficulties in integrating the two systems are due to heterogeneity and dynamicity. Heterogeneity exists in the form of different sensors and actuators, software platforms, communications, data formats and semantics. Dynamicity exists in the form of a rapidly changing environment where devices enter and leave at various points in time. In this paper, we proposed a four-layer architecture for integrating robots and smart spaces which efficiently addresses these difficulties. This architecture enables devices available in a smart space to be represented as software services. It provides applications with a homogeneous interface to heterogeneous hardware, such as sensors and actuators deployed in the smart space. Applications are automatically notified whenever new devices are introduced into the space or existing devices leave the space. Representing devices as services also allows easy modification of existing applications to enable it to make use of newly available devices. Finally, we discussed the integration of URC with the Atlas Platform, which implements this architecture and provides better services, which enhances the sentience of URC and improves physical interaction between the smart space and the users.

Acknowledgment

This research has been supported by the Korean Ministry of Information and Communication (Grant No. 2007-S-026-01), and is part of the URC Project.

References

[1] Hyun Kim, Young-Jo Cho, and Sang-Rok Oh. "CAMUS: A middleware supporting context-aware services for network-based robots", IEEE Workshop on Advanced Robotics and Its Social Impacts, Nagoya, Japan, 2005

[2] J. King, R. Bose, H. Yang, S. Pickles and A. Helal, "Atlas - A Service-Oriented Sensor Platform," Proceedings of the first IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006). Tampa, Florida, November 2006.

[3] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah, and E. Jansen, Gator Tech Smart House: a programmable pervasive space. IEEE Computer magazine, 66-74, March 2005.

[4] C. Kidd, et al., The Aware Home: A living laboratory for ubiquitous computing research. In Proceedings of Cooperative Buildings, 191-198, 1999.

[5] Hagras Et al., Creating an ambient-intelligence environment using embedded agents. IEEE Intelligent Systems, Volume 19, No. 6, 12-20, November/December 2004.

[6] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward and A. Hopper. Implementing a Sentient Computing System. IEEE Computer Magazine, Volume 34, No. 8, 50-56, August 2001.

[7] A. Chen, R.R. Muntz, S. Yuen, I. Locher, S. Park. M.B. Srivastava, "A Support Infrastructure for the Smart Kindergarten", IEEE Pervasive Computing, vol. 1, no. 2, April-June 2002, pp. 49-57

[8] T. Hansen, J. Bardram and M. Soegaard Moving out of the lab: deploying pervasive technologies in a hospital. IEEE Pervasive Computing, Volume 5, Issue 3, 24-31, July-September, 2006

[9] Matthias Kranz, Radu Bogdan Rusu, Alexis Maldonado, Michael Beetz, and Albrecht Schmidt. A Player/Stage System for Context-Aware Intelligent Environments. In Proceedings of UbiSys'06, System Support for Ubiquitous Computing Workshop, at the 8th Annual Conference on Ubiquitous Computing (Ubicomp 2006), Orange County California, September 17-21, 2006, 2006.

[10] Network Robot Forum. www.scat.or.jp/nrf/English/.

[11] Joonmyun Cho and Hyun Kim, "Context knowledge management in pervasive computing", Proceedings of the 3rd International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2006), 2006

[12] Jena – A Semantic Web Framework for Java, http://jena.sourceforge.net/, Hewlett-Packard Development Company, LP. 2005