Experience of Enhancing the Space Sensing of Networked Robots Using Atlas Service-Oriented Architecture

Sumi Helal[1], Shinyoung Lim[1], Raja Bose[1], Hen-I Yang[1], Hyun Kim[2], and Young-Jo Cho[2]

[1] Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL 32611, USA
{helal,lim,rbose,hyang}@cise.ufl.edu
[2] Intelligent Robot Research Division ETRI, Daejeon, Korea
{hyunkim,youngjo}@etri.re.kr

Abstract. In this paper, we describe how we enhance the space sensing capabilities of the Ubiquitous Robotic Companion (URC) developed by ETRI Korea, by utilizing the plug-and-play service-oriented architecture provided by University of Florida's Atlas Platform. Based on our experience in deploying services using the Atlas Platform in smart houses, for the elderly and people with special needs, a requirement emerged for utilizing networked robots to assist elderly people in their activities of daily living. Networked robots consisting of sensors, actuators and smart devices can collaborate as mobile sensing platforms with the other networked robots in a smart space, providing a complex and sophisticated actuator and human interface. This paper describes our experience in designing and implementing system architecture to integrate URC robots into pervasive computing environments using the Atlas Platform. We also show how the integrated system is able to provide better services which enhance the space sensing capabilities of URCs in the smart space.

Keywords: Ubiquitous computing, Pervasive computing, Embedded design, Service-oriented architecture, Assistive technology for elderly people.

1 Introduction

A Ubiquitous Robotic Companion (URC) is a new design concept developed at ETRI, Korea for networked robotic systems. The fundamental concept behind URC is to share the robot's three core functions – sensing, processing and actuation – over the network. URC robots typically only have sensing and actuation capabilities and outsource their processing tasks over the network to a URC server. Furthermore, URC robots use external sensors in the URC environment rather than solely depending on on-board sensors. The URC server communicates with the robots via a broadband network, and enables them to overcoming their on-board memory and processor constraints [1]. The external sensor nodes which are deployed in the environment transmit sensor data to the URC server which then provides device-specific control data to

S. Lee et al. (Eds.): APCHI 2008, LNCS 5068, pp. 1–10, 2008.

the URC robots. ETRI has been using URC technology for real-world applications since 2004, and the URC concept has been verified by the field tests. During this process, it was recognized that in order for the URC to fully utilize external sensors and actuators in the environment, it is critical to reduce the effort in configuration. The plug-and-play service oriented features of the Atlas Platform, developed at the University of Florida, offers a comprehensive solution for easy extension and integration of URC. Furthermore, when augmented by embedded sensor platforms such as Atlas, the space sensing capability of the URC robot can be dramatically improved. However, it is not easy to deploy URC robots in smart spaces, because robotic environments are diverse and subject to dynamic changes especially since, URC robots are moving around in the smart space.

The University of Florida Mobile and Pervasive Computing Laboratory has accumulated significant experience in designing and building smart spaces over the years. In particular, the technology developed has culminated in the grand opening of the Gator Tech Smart House (GTSH) project, a 2,500 sq. ft. stand-alone intelligent house designed for assistive independent living for elderly residents. The primary technical component of GTSH is the Atlas Platform [2], which includes plug-and-play Atlas sensor and actuator nodes, Atlas middleware that embraces service-oriented architecture, and a set of intuitive tools for easy configuration and fast implementation of smart spaces.

The collaboration between ETRI and the University of Florida fuses together the streamlined embedded framework of the URC robots with service-oriented architecture of the Atlas Platform. The result is a fast and easily deployable smart space where robots can collaborate with embedded sensors and each other to enhance the space sensing and processing capabilities and provide better assistant services. Moreover, the introduction of the URC into smart houses brings unforeseen capabilities for performing complex actuations on mobile platforms that can provide better assistance to the elderly, who require mobility and human interaction. In this paper, we present our experience of designing and implementing the system architecture that integrates URC robots and Atlas service-oriented architecture in the smart space to enhance space sensing capabilities of URC robots.

The remainder of the paper is organized as follows. We discuss related work in Section 2, proposed framework architecture in Section 3, implementation of the framework in Section 4 and conclusion in Section 5.

2 Related Work

There are 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 ETRI 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.

Pervasive computing technologies have opened up new opportunities for providing robots with interactive 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 in 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 devices and actuators in the same environment.

3 Proposed Framework Architecture

3.1 Experience of ATLAS in the Gator Tech Smart House

During the process of building the full-scale, 2,500 sq. ft. freestanding Gator Tech Smart House, we designed and implemented the ATLAS architecture. This architecture includes the ATLAS sensor and actuator hardware platform as the bridge between the physical and digital worlds; the ATLAS middleware, and the service authoring tool. It supports self-configuration and facilitates programmability of services in smart spaces, and the implementation has become the central core component in our smart homes.

Requirements of system support for pervasive computing environments are extremely diverse depending on the application domains. For instance, a pervasive computing system in assistive living would be vastly different from habitat monitoring in a remote forest. Since our work is primarily concerned with assistive living in smart homes, the design of the ATLAS architecture follows certain assumptions.

First, we assume that there is a light-weight centralized home server with capabilities similar to set-top boxes or access points, that has a global view of the smart home and management capabilities for the various entities and services under the roof. We also assume that the underlying network runs TCP/IP protocol, with most of the entities located inside a private network, with a NAT-enabled gateway for connections to and from any outside services and entities. The abundance of power in regular households means low power design and power management would not be a primary concern in the design.

To make ATLAS middleware the foundation of the kind of pervasive computing systems we envisioned, it is important that it should be able to fulfill the following functionalities and objectives. First of all, this middleware architecture has to be modular and extensible; it should based on a service-oriented architecture, in which each application and device is represented as a service entity, so their entrance, departure and movement can be interpreted and handled more easily; the services can utilize other software components, but they must be built in a modular fashion such that the modules can be shared and reused, and the system should be able to determine if

the required modules are present before initiating any service; the system should be easily programmable so its capability is extensible and customizable, meaning that it allows programmers to write new software while utilizing the services provided by existing modules.

Other key functionalities and objectives include the adherence to existing and upcoming standards, the provision of security, safety and privacy features and the mechanism to support scalable concurrent operations by hundreds of components.

At the bottom of the ATLAS architecture are sensor nodes with plug-and-play capabilities. The nodes connect to physical devices such as sensors and actuators, and provide a bridge between the physical world and the digital realm. The ATLAS middleware is implemented as a collection of collaborating modules. The middleware running on the central server is based on OSGi, which is an open standard that defines and provides facilities for service registry, life cycle management and dynamic binding.

On top of OSGi are several service modules in the form of software bundles. Bundle repository manages the collection of service bundles for the various devices that can connect and be integrated into the smart environment; Network Manager keeps track of all the nodes in the network and the services they provide; Configuration Manager is responsible for retrieving the various service bundles required by the nodes and for remotely configuring them over the network; Context manager and safety monitor uses a context-driven model that is independent of the active services to monitor the context changes and alerts users of the occurrence of any unsafe or undesirable context; Communication modules provide an array of standard protocols that can be used for the external communication with other business software, front-end portal, or any other collaborating system and stand-alone software. These modules can also facilitate the inter-bundle communication between various components residing in the same OSGi framework should the need arise; ATLAS Developer API provides a unified interface for programmers to interface and control diverse sensor, actuator and service entities; Service authoring tool is provided in the form of an ATLAS plug-in for Eclipse, and it provides the capability for programmers to browse through the available entities, choose the components they need, and create and deploy new services by simply specifying the application logic.

3.2 Framework Design

The collaboration between ETRI and University of Florida led to the design of a programmable sensor framework for the URC system that bridges real world sensing and actuation. The result is a fast and easily deployable smart space that networked robots can collaborate with and surrogate to, which can offer and extend its sensing and processing capabilities. On the other hand, the introduction of the URC into smart houses brings unforeseen capabilities for performing complex actuations on mobile platforms that can provide better assistance to the elderly that require mobility and human interaction. We present the design of a framework architecture which integrates URC robots with smart spaces in order to enhance the sentience of the URC.

3.2.1 Motivation

We consider sensors and actuators in the environment as unconscious robots which collaborate with software robots and physical robots. The ETRI's URC project is

regarded as one of major efforts to improve robots' intelligence using pervasive computing technologies. It enables robots to reduce cost and improve service qualities by taking full advantage of ubiquitous network environment and the networked server framework. When we tried to integrate the programmable sensor framework into the URC system, the first thing we encountered was the difference in architecture and implementation environment.

As ATLAS is implemented on top of OSGi, the integration of the framework into the URC system requires adaptation, new interfaces or change in its architecture. To resolve these issues, we had intense discussions with the ETRI team on the design of the framework with minimum implementation effort and modification in each system's architecture.

3.2.2 Functions in the Framework

To integrate the ATLAS service-oriented middleware into the URC's middleware, *i.e.*, Context-Aware Middleware for URC Systems (CAMUS), we discussed two different approaches from the perspective of architecture; (1) Comply with the ETRI URC architecture and (2) Build interface of OSGi in the ETRI URC side. The task of CAMUS is to connect and control 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. Additional functions including voice recognition, voice synthesis and image recognition are heretofore executed in a robot itself.

Figure 1 shows a conceptual architecture based on the ETRI URC architecture. We found that the architecture in Figure 1 would give two different architectures with considerable changes in each other's architecture for integrating the programmable sensor framework into the URC system. For instance, if we design the system that operates on top of OSGi, the OSGi will serve as a bridge between the two different architectures at the cost of the redundancies and seemingly influences possible delay in response time.

For optimized design of the framework, we found out that the CAMUS [1] consists of three main modules: Service Agent Managers, the CAMUS server and the 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. And we also found that the Planet is the communication framework between the SAMs and the CAMUS server.

Based on this architectural information and our discussions, we decided to integrate the ATLAS service-oriented middleware with the architecture of SAM. The *service agent* is a software module which acts as a proxy to connect various sensors and devices to the CAMUS server. It raises software events to notify the CAMUS server that its device detects some noticeable changes and receives requests from the CAMUS server to actuate its device. The SAM is a software container of these service agents that resides on a robot or in a space, and manages them. SAM plays a central role in the integration of the ATLAS and the CAMUS systems. The collaboration between smart spaces and robots are facilitated by the communication between the ATLAS middleware and the SAM.

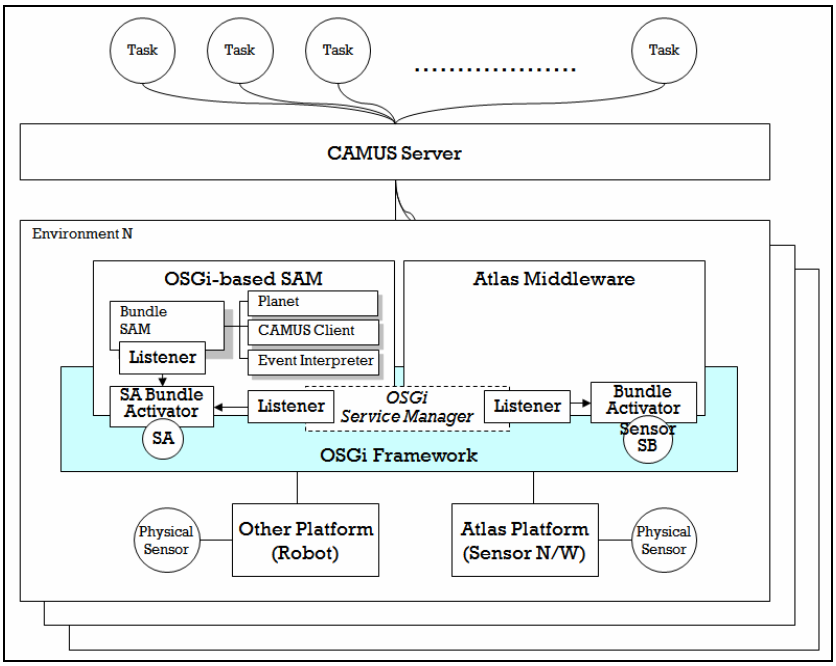

Fig. 1. Conceptual architecture of the ETRI URC based architecture

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 the ATLAS Middleware and registers this service to the CAMUS server so that the 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.

In the updated architecture of the SAM, the ATLAS service-oriented middleware is being monitored by the *Listener* in the *Service Agent Loader*. When the sensing event happens in each *Bundle Activator*, then the *Listener* sends an activation signal to the *Sensor Interpreter*. The *Sensor Interpreter* then, receives sensed data from the *Bundle Activator*. After the CAMUS sends commands information regarding the situational response of the sensed data, the *Service Agent Invocator* transmits it to the *Bundle Activator* to activate the ATLAS platform to run the target actuator. Therefore, the ATLAS service-oriented middleware has communication channels with the *Listener*, the *Service Interpreter* and the *Service Agent Invocator* to each *Bundle Activator*.

The integration between the systems requires service bundles to implement interfaces to communicate with the other service in the ATLAS middleware, as well as the 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 the CAMUS server to invoke methods at service agents. A key feature of the Service Agent Invocator is supporting

the asynchronous method invocation. It participates in necessary scheduling and threads management activities for 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 the SAM. The sensor interpreter examines each event, passing, dropping, refining or aggregating them. Sometimes the Event Interpreter inserts new events based on the event it examines. By this way, any duplicated or unnecessary events are filtered out at the SAM, reducing network and computational overhead at the 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 the CAMUS server. Any subscriber to the queue then receives notification of that event.

4 Implementation

The service oriented architecture of Atlas is hosted on an OSGi framework. The URC framework called CAMUS (Context Aware Middleware for URC Systems) from ETRI on the other hand runs outside OSGi and does not come with OSGi based components. Hence, the main task of this integration effort was to create a bridge between the two systems in a manner which fully utilizes both, the plug-and-play features of Atlas and the context-aware backend provided by CAMUS.

4.1 Integration Process

The Atlas Platform represents every sensor connected to it as an OSGi service object. Regardless of a sensor's type and manufacturer, Atlas provides a uniform interface (declared as a Java interface called *AtlasService*) to every sensor. This ensures that application developers are abstracted away from low-level details and variations between different sensors and can use a single set of high-level methods to access multiple heterogeneous sensors.

The CAMUS is responsible for acquiring sensing information from robots and environments, planning tasks based on the current context and sending control messages to robots. The Service Agent Manager (SAM) is the robot-side component of CAMUS which sends a robot's sensing data to the CAMUS server and receives control commands from the CAMUS server. As described earlier, it was decided to integrate by modifying SAM to become the link between the two systems.

First, SAM was ported to OSGi to enable it to run as a bundle inside the service framework. The OSGi version of SAM is split into two parts: The *Service Agent Loader* component and data acquisition components. The data acquisition components implement the *AtlasService* interface to enable them to become Atlas device bundles which can represent hardware devices connected to the Atlas Platform. Hence, as shown in Figure 2, SAM implements interfaces for both CAMUS and Atlas and bridges the connection between the two.

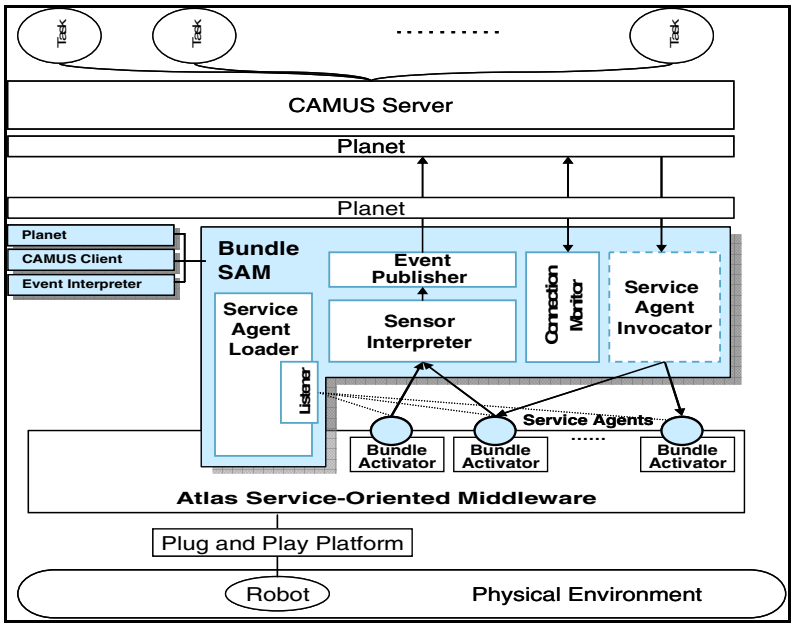

Fig. 2. Integrating ATLAS with CAMUS

Implementing the *AtlasService* interface enables SAM to become an integral part of a sensor service. In our example below, SAM components are present as part of a Pressure Sensor bundle, which enables them to be automatically loaded by Atlas whenever the hardware sensor is powered on. This bundle on being loaded also establishes connection with the *Service Agent Loader* and is able to send data to the CAMUS server and receive commands from it.

4.2 System Execution

When a new device is deployed in the space, the plug-and-play capability of Atlas ensures that a service bundle representing that sensor or actuator is automatically activated in the OSGi framework. The service bundle also contains the data acquisition components of SAM, which automatically seek out the service agent loader using native OSGi calls. The *Service Agent Loader* then registers this service to the CAMUS server so that CAMUS tasks can use this service. Similarly, when an existing device is removed from the space, the corresponding service bundle automatically expires and the service agent loader on detecting this event 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. 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 then receives the event notification.

5 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 multiple devices can enter or 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.

Acknowledgments. This paper has been supported by the Korean Ministry of Information and Communication (Grant No. 2007-S-026-01) which is a part of ETRI's URC research project.

References

1. Kim, H., Cho, Y.-J., Oh, S.-R.: CAMUS: A middleware supporting context-aware services for network-based robots. In: IEEE Workshop on Advanced Robotics and Its Social Impacts, Nagoya, Japan (2005)
2. King, J., Bose, R., Yang, H., Pickles, S., Helal, A.: Atlas - A Service-Oriented Sensor Platform. In: Proceedings of the first IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006), Tampa, Florida (2006)

3. Helal, A., Mann, W., Elzabadani, H., King, J., Kaddourah, Y., Jansen, E.: Gator Tech Smart House: a programmable pervasive space. IEEE Computer magazine, 66–74 (March, 2005)
4. Kidd, C., et al.: The Aware Home: A living laboratory for ubiquitous computing research. In: Proceedings of Cooperative Buildings, pp. 191–198 (1999)
5. Hagras, et al.: Creating an ambient-intelligence environment using embedded agents. IEEE Intelligent Systems 19(6), 12–20 (2004)
6. Addlesee, M., Curwen, R., Hodges, S., Newman, J., Steggles, P., Ward, A., Hopper, A.: Implementing a Sentient Computing System. IEEE Computer Magazine 34(8), 50–56 (2001)
7. Chen, A., Muntz, R.R., Yuen, S., Locher, I., Park, S., Srivastava, M.B.: A Support Infrastructure for the Smart Kindergarten. IEEE Pervasive Computing 1(2), 49–57 (2002)
8. Hansen, T., Bardram, J., Soegaard, M.: Moving out of the lab: deploying pervasive technologies in a hospital. IEEE Pervasive Computing 5(3), 24–31 (2006)
9. Kranz, M., Rusu, R.B., Maldonado, A., Beetz, M., Schmidt, A.: A Player/Stage System for Context-Aware Intelligent Environments. In: Dourish, P., Friday, A. (eds.) UbiComp 2006. LNCS, vol. 4206, Springer, Heidelberg (2006)
10. Network Robot Forum, http://www.scat.or.jp/nrf/English/