International Journal of Fuzzy Logic and Intelligent Systems, vol. 8, no. 4, December 2008 pp. 288-298

Networked Robots using ATLAS Service-Oriented Architecture in the Smart Spaces

Sumi Helal*, Raja Bose*, Shinyoung Lim† and Hyun Kim*****

*** University of Florida**
**** University of Texas at Dallas**
***** ETRI**

Abstract

We introduce new type of networked robot, Ubiquitous Robotic Companion (URC), embedded with ATLAS Service-oriented architecture for enhancing the space sensing capability. URC is a network-based robotic system developed by ETRI. For years of experience in deploying service with ATLAS sensor platform for elder and people with special needs in smart houses, we need networked robots to assist elder people in their successful daily living. Recently, pervasive computing technologies reveals possibilities of networked robots in smart spaces, consist of sensors, actuators and smart devices can collaborate with the other networked robot as a mobile sensing platform, a complex and sophisticated actuator and a human interface. This paper provides our experience in designing and implementing system architecture to integrate URC robots in pervasive computing environments using the University of Florida's ATLAS service-oriented architecture. In this paper, we focus on the integrated framework architecture of URC embedded with ATLAS platform. We show how the integrated URC system is enabled to provide better services which enhance the space sensing of URC in the smart space by applying service-oriented architecture characterized as flexibility in adding or deleting service components of Ubiquitous Robotic Companion.

Key Words : pervasive computing, service-oriented architecture, assistive technology

1. Introduction

A Ubiquitous Robotic Companion (URC) is a new concept for a networked robotic system. The motivation of a URC stems from the reduction of expensive cost of a fully equipped intelligent robot system which comes with all the required technical features within a system. The basic concept of URC is to share the robot's three core technical features – sensing, processing and action – on the network as shown in figure 1. It means that URC robots may have minimum sensing and action capabilities to communicate with URC server on the network to acquire processing capability. They use external sensors in the URC environment rather than embedding sensors into the URC robots. When they use URC server, which is connected via a broadband network, for overcoming their on-board memory and processor constraints, the external sensor nodes transmit sensed data to the URC server and the URC robots receive control data of specific device from the URC server [8, 18]. ETRI has developed URC technology and applied it to URC services in real world since 2004.

After verification of URC concept by field tests, the authors decided that the URC should fully utilize external sensors and actuators in the environment by adopting UF's ATLAS service-oriented architecture. If the URC's sensing features be enhanced by using embedded sensor platforms, the space sensing capability of the URC robot would be dramatically improved.

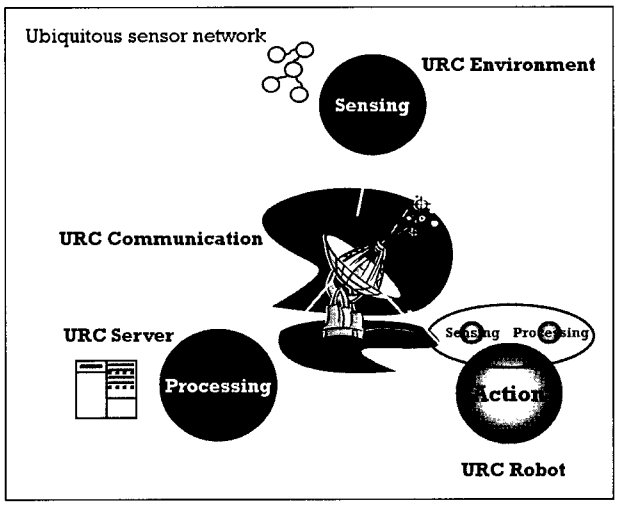

Fig. 1. URC's Three Core Technical Features

However, it is not easy to deploy URC robots in smart space, because robot environments are diverse and subject to change dynamically and even URC robots are moving around in the smart spaces.

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

Manuscript received Oct. 6, 2008; revised Dec. 10, 2008.
†Corresponding Author

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 [12, 19], 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 configurations and fast implementation of smart spaces.

The collaboration between ETRI and UF brings together the streamlined embedded framework of the URC robots and ATLAS service-oriented architecture. The outcome of the collaboration is a fast and easily deployable smart space that robots can collaborate with embedded ATLAS sensor platform to assist elder people and to co-work with other URC robots which enhance space sensing and request of processing capabilities. On the other hand, the presence of the URC into smart houses brings unforeseen capabilities to perform high order of complex actuations on mobile platforms that can provide better assistance to the elderly that require mobility and social communications. In this paper, we present our experience of design and implement of the URC system with ATLAS architecture that integrates the URC robots and the ATLAS service-oriented architecture in the smart space to enhance the space sensing of the URC robots for successful independent living of elderly and disabled people in smart houses.

The remainder of the paper is organized as follows. We discuss related works in Section 2, experience of Gator Tech Smart House in Section 3, proposed framework architecture in Section 4, implementation of the framework in Section 5 and conclusions in Section 6.

2. Related Work

There are a number of 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 [22]. The networked robot project of the Japanese Network Robot Forum [34] 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. Various types of robots had been investigated through bridging between robots and ubiquitous networks. Through collaboration and interactions among robots, multifaceted and diversified services were also investigated. The ETRI URC project also is one of major efforts to improve robots' intelligence using pervasive computing technologies [18]. It enables robots to reduce costs and improve quality of services by taking full advantage of ubiquitous network environment and the networked server framework.

The pervasive computing technologies have opened new possibilities of 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 [12, 13, 17, 19], offices [1, 9, 15, 20], factory [24, 26, 27], open space [21], classrooms [2], and hospitals [10, 14, 31]. 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 [5, 16, 23, 25]. 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.

3. Experience of Gator Tech Smart House

3.1. Experience in Early Stage

The goals and requirements of the proposed framework architecture were conceived during our recent work in prototyping "assistive environments" for elder people and individuals with special needs. Housed inside the pervasive computing lab and occupying over 500 sq ft, Matilda Smart House was the first attempt at creating such assistive environments at the University of Florida. We had no guiding architecture while deploying the Matilda Smart House. We integrated tens of sensors, actuators, appliances and other components including contact sensors, motion sensors, cameras, ultrasonic transceivers, X10 modules and controllers, several microprocessor-based controllers, a microwave oven, an entertainment system, mobile phones, and a home PC. The Matilda Smart House fulfilled its purpose as a demo platform, but more importantly, it solidifies the true need for architecture and helps in identifying its goals and requirements, which we summarize below:

- We wished to eliminate the need for system integration. We wanted it to be possible for an entity joining the space to self-explore and self-integrate itself. We also wanted the pervasive space to be able to cleanly remove exiting (or failed) entities. This plug and play wish was highest on our list.
- We wished to decouple application development from the physical world of sensors and actuators. Any hard-coding of physical artifacts within the applications should be avoided. To achieve this, we wanted any physical entity (sensor or actuator) to be liquidated into a basic software service at the time (and as a result) of self integration. This way we would

no longer deal with physical entities and instead deal with their service representations.

- We wished to program the pervasive space. Capitalizing on the service-oriented view of sensors and actuators, we allowed ourselves to wish for a pervasive space that would be able to map itself automatically into a software development project within an integrated development environment (IDE) such as Visual Studio or Eclipse. This big wish was equivalent to asking for total control over the management, configuration and applications development of pervasive computing systems. We knew that if this would ever be possible, we would have completely changed the skill set needed to build pervasive spaces. Instead of engineers and costly system integrators, affordable and highly available computer programmers with standard Java or .NET skills would be all that is needed to develop and program pervasive spaces.
- We wished for the pervasive space to be open and flexible to embrace a variety of entities without any special favor towards particular participants or their underlying technology. Well-accepted standards should be utilized.

3.2. Experience of ATLAS in Gator Tech Smart House

To make the items in the wish list above come true, during the process of implementing 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 platform as the basic 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 piece 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 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 [3, 4, 6, 7, 11, 28, 29, 30, 32, 33]. 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.

ATLAS must also handle the dynamicity and heterogeneity in a typical pervasive computing environment; the hardware and software components should support the notion of plug-and-play, where components are able to self-configure and self-organize when joining the system, the drivers and communications protocols should be setup automatically as soon as they join, and the other existing software modules can discover and use these new components.

The capability to dynamically manage the life cycle of each component is also crucial, because the openness and dynamic nature of smart environment, it is common for new entities to enter, and existing entities to be replaced or went bad, the middleware needs to handle these events gracefully, providing smooth introduction, modification, replacement, or termination; the middleware should also provide an API for interacting with and accessing diverse devices, so the consumers of the heterogeneous devices and services can have a consistent way to view and manipulate them.

Other main 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. With these assumptions and functional requirements, we designed the ATLAS architecture as shown in figure 2.

At the bottom of the ATLAS architecture is the ATLAS nodes with plug-and-play capabilities, the nodes connect to physical devices such as various sensors and actuators, and provides a bridge for communicating 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.

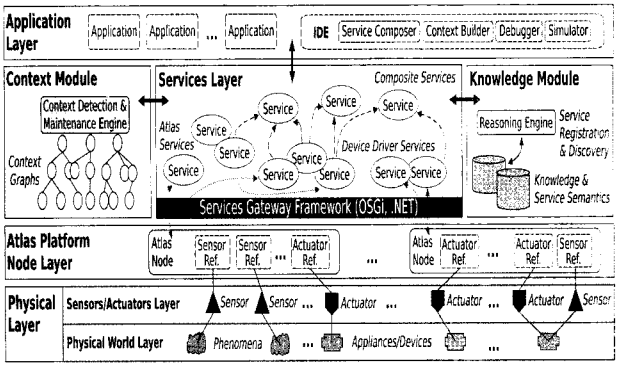

Fig. 2. ATLAS Architecture

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 to 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.

4. Proposed Framework Architecture

4.1. Framework Design

The collaboration between ETRI and UF brings together discussing the design of the programmable sensor framework for the URC system that bridges real world sensing and actuating by way of the URC platform. 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 to perform high order of complex actuations on mobile platforms that can provide better assistance to the elderly that require mobility and social communications. In this section, we present the motivation and design of integrated framework architecture to merge the URC robots into the smart space in order to enhance the sentience of the URC for the elderly and disabled people at home.

4.2. 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 costs and improve quality of service by taking full advantage of ubiquitous network environment and the networked server framework. When we try to integrate the programmable sensor framework into the URC system, the first thing we have encountered is the difference in architecture and implementation environment.

Figure 3 describes conceptual network configuration showing the two different systems communicate each other with the ATLAS middleware and CAMUS-PLANET. As the ATLAS is implemented upon the OSGi, the integration of the framework into the URC system requires adaptation, interfaces or change in its architecture. We have discussed with the ETRI team on the design of the framework with minimum efforts of implementation and changes in each other's architecture.

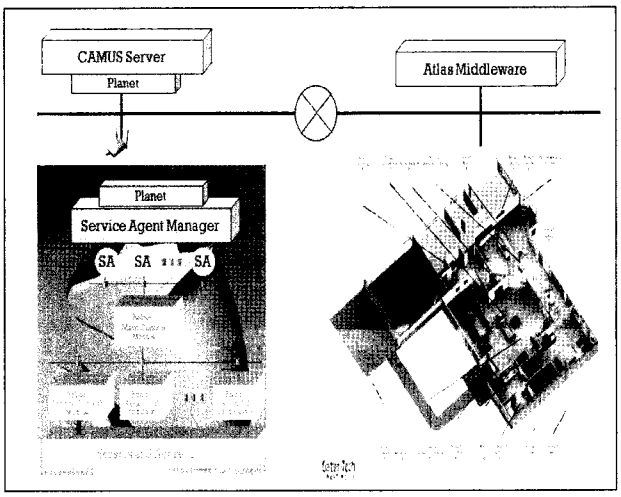

Fig. 3. Conceptual network configuration of two different systems

4.3. 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 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 control and actuation 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. Figure 4 is conceptual architecture based on the ETRI URC architecture.

We found that the architecture in figure 4 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 be a bridge of the two different architectures but it generates redundancies and seemingly influences system performance, for instance, possible delay in response time.

For optimized design of the framework, we find 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 and actuation 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 find that the Planet is the communication framework between the SAMs and the CAMUS server.

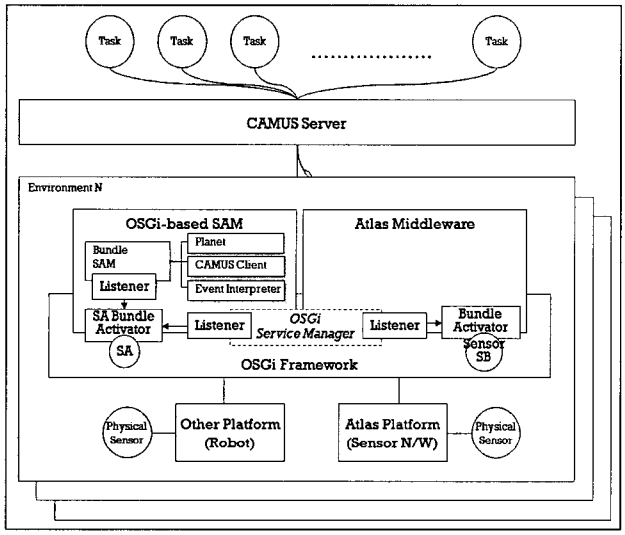

Fig. 4. Conceptual Architecture of the ETRI URC

With this URC's architectural information and discussions, we integrate the ATLAS service-oriented middleware in the architecture of the SAM. The service agent (SA in figure 4) is a software module performed 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 reside in a robot or 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. Figure 5 shows the updated architecture of the SAM that holds the ATLAS service-oriented middleware module.

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 it 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.

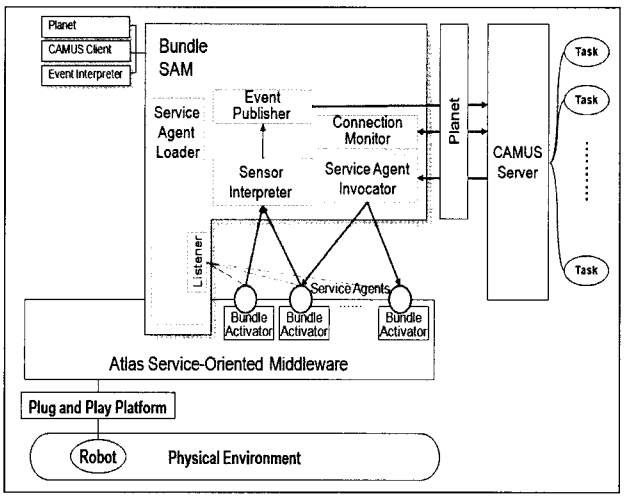

Fig. 5. Updated Architecture of the SAM

As shown in the figure 5, 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 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 takes parts in necessary scheduling and threads 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 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 will be received it.

5. 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.

5.1. 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 following source code is an example of interfaces between ATLAS Development Environment and SAM which is an interface specifying methods available for accessing sensors via ATLAS. The source code depicts 'InterlinkPressureSensor' service for the service representation of the Force Sensing Resistor based pressure sensor made by Interlink.

```
package org.sensorplatform.sensors.pressure;
import com.pervasa.atlas.dev.service.AtlasClient;

/** Interface Definition **/
public interface InterlinkPressureSensor
{
/** Poll Sensor to get reading (Pull)**/
public void getPressureReading(AtlasClient ac);

/** Ask Sensor to send stream of readings (Push) **/
public void subscribeToPressureData(AtlasClient ac);

/** Ask Sensor to stop sending stream of readings **/
public void unsubscribeFromPressureData(AtlasClient ac);

/** Get the system time when this sensor came online **/
public long getStartupTimeMillis();
}
```

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. 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.

The following source code is an example of SAM running inside of OSGi with SAs, as one of the ATLAS Device Service Bundles, to activate the sensor service bundles, which is dynamically executed when the sensor comes online. The instance of this bundle is loaded on OSGi to act as its service representation.

```
// Reference to the OSGi framework bundle context
private BundleContext context;
//Location of service bundle,
//ID of Atlas Node to which sensor is connected
 String bundleLocation, nodeId = new String("");
/*Start this service (called by OSGi when bundle is installed)*/
 public void start(BundleContext context) throws Exception
  {
            try
            {
             bundleLocation = (String)
```

```
(context.getBundle().getHeaders()).get("Bundle-Location");
                StringTokenizer st =
      new StringTokenizer(bundleLocation,"-");
                nodeId = st.nextToken();
               }catch(Exception e)
                { }
//Start Atlas part of interface
//Register Sensor as an Atlas Service in the OSGi Framework
                final String[] c = { AtlasService.class.getName() };
                final Properties p = new Properties();
                p.put("Node-Id", nodeId);
                this.context = context;
                m_service = new AtlasPressureImpl(null, null);
                context.registerService(c, m_service, p);
//Log Error if service cannot be instantiated
                        if(m_service == null)
    {
                         if (logger.isInfoEnabled())
     {
                         logger.info("start(BundleContext):    Not
Found
        InterlinkPressureSensor Service.");
                         }
                         return;
                        }
//Start CAMUS part of interface
                        ServiceBeanInfo      info      =      new
ServiceBeanInfo(ID, NAME,
  SERVICE,                              SERVICE_PROVIDER,
SERVICE_PROVIDER_IMPL,
                          null,m_ac, null);
//Listen for Service Agent Loader bundle component of SAM
 m_serviceListener = new SamBundleEventListener( context);
                        ServiceReference re=
 context.getServiceReference(Maintainable.class.getName());
 Maintainable m = (Maintainable)context.getService(re);
//Add this sensor service as a service agent
    m.addServiceAgent(info);
    context.addServiceListener(
    m_serviceListener,"(objectClass=" +
    Maintainable.class.getName() + "*)");
                        context.ungetService(re);
            }
/* Called by OSGi framework when bundle is unloaded */
            public  void  stop(BundleContext  context)  throws
Exception
  {
                        ServiceReference re=
 context.getServiceReference(Maintainable.class.getName());
//Remove this sensor service as an available service agent
   Maintainable m = (Maintainable)context.getService(re);
                        m.removeServiceAgent(ID);
  context.removeServiceListener(m_serviceListener);
//Clean up this sensor service
                        m_service.close();
            }
}
```

Hence, as shown in Fig. 6, SAM implements interfaces for both CAMUS and ATLAS and bridges the connection between the two. Implementing the AtlasService interface enables SAM to become 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.

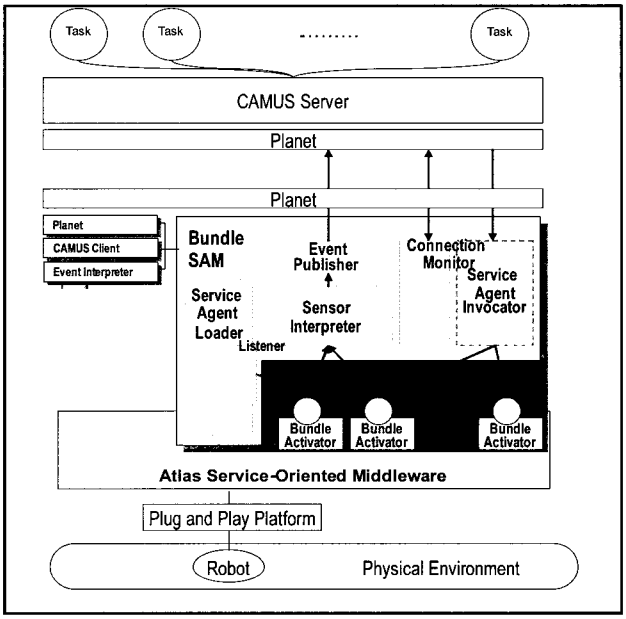

Fig. 6. Integrating ATLAS with CAMUS

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.

5.2. System Execution

After we implemented the URC platform with ATLAS integration, we executed the system based on the service scenario of adding/deleting sensors. We describe series of ooperations of adding/deleting sensors on the URC platform as shown in figure 7.

The whole process of adding/deleting sensors on the URC platform can be described in the form of protocol including each different phases. Here, we describe the protocol with entity, data and data transfer with condition in narrative method.

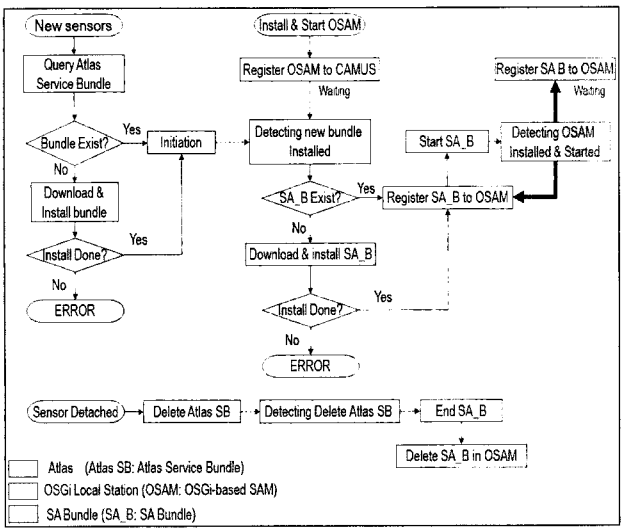

Fig. 7. Operations of Adding/Deleting sensors on URC Platform

● ***Phase-1: New sensors attached to the URC platform***

When users or service person are attaching sensors or actuators to the URC platform, the URC platform will find new devices are attached. Then, the URC platform queries the OSGi-based SAM in the CAMUS server to find the appropriate ATLAS service bundle.

● ***Phase-2: Query the ATLAS Service Bundle***

OSGi-based SAM receives the query from the URC platform, and then the SAM sends the query to the OSGi bundle repository.

● ***Phase-3: Installing the ATLAS Service Bundle***

If there is the bundle in the OSGi bundle repository, then, the SAM initiates the bundle. Or if there is no bundle available in the OSGi bundle repository, the SAM downloads and installs the bundle from the external source.

● ***Phase-4: Installing the OSGi-based SAM (Manually)***

For OSGi-based SAM service, the CAMUS server installs and starts the OSGi-based SAM in the CAMUS server system. When the CAMUS server starts the OSGi-based SAM, the Listener module in the OSGi-based SAM listens and waits for the signal from the URC platform.

● ***Phase-5: URC platform installs the new sensor's bundle***

When the URC platform installs the bundle, the URC platform sends the signal to the CAMUS server. When the Listener module in the OSGi-based SAM receives the sensed data from the ATLAS platform, the SAM sends it to the CAMUS server and waits for the response from the CAMUS server. And when the CAMUS server sends the control commands information, the SAM sends query of Service Agent Bundle for the commands information.

● ***Phase-6: Installing the Service Agent bundle(actuators)***

If there is the Service Agent Bundle, then, the OSGi-based SAM installs the bundle for the actuators. If there is no bundle available, the OSGi-based SAM downloads and installs the Service Agent Bundle from the external source.

● ***Phase-7: Register Service Agent Bundle (Manually)***

Along with adding the new sensors to the URC platform, the developers program the Service Agent Bundle for the service through actuators and register to the OSGi-based SAM manually. And then, the Listener module in the SAM listens and waits for the newly added sensors.

● ***Phase-8: Start the Service Agent Bundle***

When the OSGi-based SAM, i.e., the Listener module, receives commands information from the CAMUS server, the OSGi-based SAM starts the Service Agent Bundle to the URC platform where the appropriate actuator for the service is connected to the ATLAS platform.

Figure 8 is the framework diagrams describing the simple protocols with two different communication paths of sensing and commands information in figure 7. As there are two different communication paths in the framework, the ATLAS platforms are able to communicate either with URC platform or with CAMUS server.

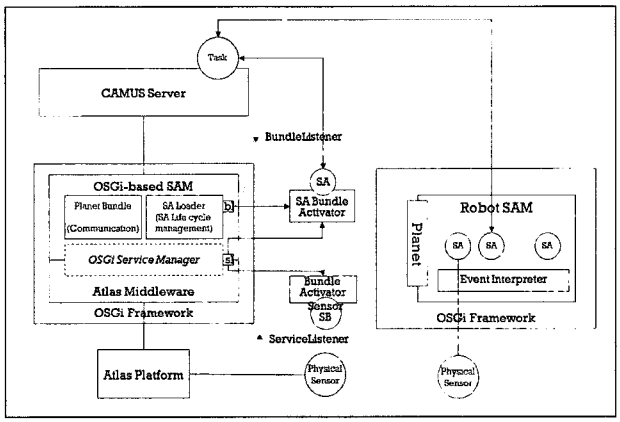

Fig. 8. Framework Diagrams of Adding/Deleting Sensors on URC Platform

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 following source code is an example of service implementation code of the sensor and contains methods which bridge the ATLAS and CAMUS systemn to ensure data flows seamlessly between them. The source code is a simple example of programmable toolkit for URC applicatuion developer.

```
{
// Logger for this class
        private static final Logger logger =
    Logger.getLogger(AtlasPressureImpl.class);
        private volatile boolean m_isStarted;
        private volatile ServiceContext m_context;
        private long startupTS;
        public AtlasPressureImpl(ServiceContext context,
Properties params)
  {
                m_context = context;
                m_isStarted = false;
//store properties associated with this sensor service
addProperty("measure-type", "pressure");
                addProperty("data-type", "int");
                startupTS = System.currentTimeMillis();
                this.isSubscribe = false;

                System.out.println("Subscribe pressure
sensor for
        logging.");
//Ask Atlas node to start pushing data stream from sensor
    subscribe(this);
        }
/* Activate Service Bean for service agent */
        public void activate()
  {
                m_isStarted=true;
                if(m_isStarted){
                        if (logger.isInfoEnabled()) {

        logger.info("AtlasPressure.startServiceBean():
      Already Started");
                        }
                }
                else{
                        m_isStarted=true;
                        if (logger.isInfoEnabled()) {

        logger.info("AtlasPressure.startServiceBean():
        Active");
                        }
                }
        }
/* De-activate Service Bean for service agent */
        public void deactivate() {
                if(m_isStarted){
                        m_isStarted=false;
                }
                else{
                        if (logger.isInfoEnabled()) {

    logger.info("AtlasPressure.stopServiceBean():
    Already Stopped");
                        }
                }
        }
```

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 then receives the event.

6. 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.

References

[1] 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.

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

[3] DEY, A., and ABOWD, G., "*The Context Toolkit: aiding the development of context-aware applications,*" Workshop on Software Engineering for Wearable and Pervasive Computing (June 2000), Limerick, Ireland.

[4] EDMONDS, N., STARK, D., and DAVIS, J. MASS, "*Modular architecture for sensor systems,*" In 4th Int'l Conf. on Information Processing in Sensor Networks, April 2005.

[5] GELLERSON, H., KORTUEM, G. SCHMIDT, A., and BEIGL, M., "*Physical prototyping with Smart-Its,*" IEEE Pervasive Computing, 3, 3, pp. 74-82, July-Sept. 2004.

[6] S. Giroux, A. Ayers, and H. Pigot, "*An Infrastructure for Assisted Cognition and Telemonitoring,*" ICADI, 2003.

[7] GREENBERG, S., and FITCHETT, C., "*Phidgets: easy development of physical interfaces through physical widgets,*" In 14th ACM Symposium on User Interface Software and Technology, pp. 209-218, Nov. 2001.

[8] 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.

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

[10] 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

[11] HARBIRD, R., HAILES, S., and MASCOLO, C., "*Adaptive resource discovery for ubiquitous computing,*" Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, pp. 155-160, 2004.

[12] 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.

[13] HUEBSCHER, M., and MCCANN J., "*Adaptive middleware for context-aware applications in smart-homes,*" Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, pp. 111-116, 2004.

[14] Joseph McDonnell, *et. al.*, "*The Cost of Treatment of Alzheimer's Disease in the Netherlands – A Regression-Based Simulation Model,*" pp. 379-390, Vol. 19, No. 4, Pharmacoeconomics 2001.

[15] Y. Kaddoura, J. King and A. Helal, "*Cost-precision tradeoffs in unencumbered floor-based indoor location tracking,*" Proc. of the 3rd Intl. Conf. on Smart homes and health Telemetrics, July 2005.

[16] KAHN, J., KATZ, R., and PISTER, K., "*Next century challenges: mobile networking for "Smart Dust",*" Proceedings of 5th ACM/IEEE Int'l Conf. on Mobile computing and networking, pp. 271-278, 1999.

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

[18] 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

[19] 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.

[20] LYMBEROPOULOS, D., and SAVVIDES, A., "*XYZ: a motion-enabled power aware sensor node platform for distributed sensor network applications,*" In 4th Int'l Conf. on Information Processing in Sensor Networks, April 2005.

[21] MAINWARING, A., POLASTRE, J., SZEWCZYK, R., CULLER, D., and ANDERSON, J., "*Wireless sensor networks for habitat monitoring,*" In 1st ACM Intl. Workshop on Wireless Sensor Networks and Apps., pp. 88-97, Sept. 2002.

[22] 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), September 17-21, 2006.

[23] MODAHL, M., AGARWALLA, B., ABOWD, G. D., RAMACHANDRAN, U., and SAPONAS, T., "*Toward a standard ubiquitous computing framework*," Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, pp. 135-139, 2004.

[24] NACHMAN, L., KLING, R., HUANG, J., and HUMMEL, V., "*The Intel mote platform: a Bluetooth-based sensor network for industrial monitoring*," In 4th Int'l Conf. on Information Processing in Sensor Networks, April 2005.

[25] O'FLYNN, B. *et. al.*, "*The development of a novel minaturized modular platform for wireless sensor networks*," In 4th Intl. Conference on Information Processing in Sensor Networks, April 2005.

[26] PARK, C., LIU, J., and CHOU, P., "*Eco: an ultra-compacy low-power wireless sensor node for real-time motion monitoring*," In 4th Int'l Conf. on Information Processing in Sensor Networks, April 2005.

[27] PON, R. *et. al.*, "*Networked infomechanical systems: a mobile embedded networked sensor platform*," In 4th Int'l Conf. on Information Processing in Sensor Networks, April 2005.

[28] ROBINSON, J., WAKEMAN, I., and OWEN, T., "*Scooby: middleware for service composition in pervasive computing*," Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, pp. 161-166, 2004.

[29] ROMÁN, M., HESS, C., CERQUEIRA, R., RANGANATHAN, A., CAMPBELL, R., and NAHRSTEDT, K., "*Gaia: a middleware infrastructure to enable active spaces*," IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.

[30] SAMIMI, F., MCKINLEY, P., SADJADI, S., and GE, P., "*Kernel-middleware interaction to support adaptation in pervasive computing environments*," Proceedings of the 2nd workshop on Middleware for pervasive and adhoc computing, pp. 140-145, 2004.

[31] A. Sixsmith, "*Pervasive Computing and the Well-Being of Older Persons*," ICADI, 2003.

[32] SØRENSEN, C., WU, M., SIVAHARAN, T., BLAIR, G., OKANDA, P., FRIDAY, A., and DURAN-LIMON, H., "*A context-aware middleware for applications in mobile ad hoc environments*," Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, pp. 107-110, 2004.

[33] YAU, S., KARIM, F., WANG, Y., WANG, B., and GUPTA, S., "*Reconfigurable context-sensitive middleware for pervasive computing*," IEEE Pervasive Computing, pp. 33-40, Jul-Sep 2002.

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

Sumi Helal

He received PhD degree in Computer Science from Purdue University in 1991. He is a professor in the Computer & Information Science & Eng. (CISE) Department at the University of Florida and a director of Mobile and Pervasive Computing Lab. His research interest includes sensor network, pervasive computing, mobile computing and intelligent robot systems for the elderly and disabled people.

Raja Bose

He received PhD degree in Computer Science from the University of Florida in 2008. He is a research scientist at Nokia Research Center Palo Alto. His research interest covers sensor platform architecture, sensor networking, programmable environment, and scalable query processing in service-oriented sensor networks into pervasive computing systems

Shinyoung Lim

He received PhD degree in Computer Science from Korea University in 2001. With his 23 years experience of Systems Engineering Research Institute (SERI), Electronics and Telecommunications Research Institute (ETRI), University of Florida, and Southern Methodist University, he is now a research faculty at the University of Texas at Dallas. His research interest includes artificial intelligence, pervasive computing, and information security for healthcare and rehabilitation of senior and people with special needs.

Hyun Kim

He received BS, MS, and PhD degrees in the Department of Mechanical Design and Manufacturing from Hanyang University, Seoul, Korea, in 1984, 1987, and 1997, respectively. He worked for Systems Engineering Research Institute (SERI) from 1990 to 1998. Since he joined ETRI in 1998, he has been a project leader in the Intelligent Robot Research Division. His research interests include networked robots, context-awareness and ubiquitous computing, distributed computing, and virtual engineering.