Building Plug-and-Play Smart Homes Using the Atlas Platform[1]

Raja Bose, Jeffrey King, Steven Pickles, Hicham Elzabadani, Abdelsalam (Sumi) Helal
Mobile & Pervasive Computing Laboratory, CISE Department
University of Florida, Gainesville, FL 32611, USA
http://www.icta.ufl.edu

{rbose, jck, spickles, hme, helal}@cise.ufl.edu

Abstract. Pervasive computing environments such as smart homes require a mechanism to easily integrate and manage numerous and heterogeneous sensors and actuators into the system. Additionally, the smart space should be assembled programmatically by software developers instead of hardwired by engineers and system integrators. However, most available sensor platforms are not adequate for this task. Atlas is a new, commercially available sensor and actuator platform that facilitates self-integration and pervasive space programmability. It allows for cost-effective development, enables extensibility, and simplifies change management in smart spaces. This paper gives a brief overview of the Atlas platform and describes how to use and program Atlas to build a smart space. The paper also presents several use cases of Atlas in the Gator Tech Smart House -- a real-life smart home dedicated to successful aging and independent living research.

Keywords. Smart spaces; smart home; pervasive computing; pervasive computing middleware; ubiquitous computing; sensor; sensor platform; actuator; actuator platform

1. Introduction

Smart homes and other pervasive computing environments require integrating a large number of sensors, actuators, and computation devices with a rich set of services providing a variety of domain-specific applications. For most of the first-generation pervasive space prototypes in existence now, this system integration task is a laborious, ad-hoc process. Adding each new device into the space requires a great deal of work. After the initial decision over which particular component to purchase, the smart space developers must research that device's characteristics and operation, determining how to configure it and interface with it. Then the device must be connected and physically integrated into the space. Any application that will use the new device must be written with specific knowledge of the resources assigned to connect the device, signals to query and control the device, and the meaning of any signals returned. This "hardwired" integration requires tedious and repeated testing to guard against errors or indeterminate behavior that could arise from conflicting devices or resources.

A typical smart house will contain far more components than could be connected to the I/O ports of a single computer, so these spaces make use of a sensor network platform (e.g., Berkeley Motes [1]) to integrate the numerous heterogeneous devices. However, while these networks expand the number of operating devices, none of the traditional sensor network platforms available today are fully adequate for developing smart spaces. Most sensor network platforms, as the name implies, focus only on sensors, ignoring actuators. Yet the actuation of devices is critical in a space such as a smart home. Some platforms (e.g., Phidgets

[1]This is a publication of the Rehabilitation Engineering and Research Center on Technology for Successful Aging, funded by the National Institute on Disability and Rehabilitation Research of the Department of Education under grant number H133E010106. The opinions contained in this publication are those of the grantee and do not necessarily reflect those of the Department of Education.

[2]) do support actuators, but are constrained by scalability issues and a fixed hardware configuration. Additionally, none of the platforms have the capability to represent automatically their connected devices as software services to programmers and users. Instead, programmers must write distributed applications that query hard-coded resources to access the devices connected to the platform. This does nothing to alleviate the "hardwired" system integration problems mentioned above.

Developing service-rich, robust smart spaces requires a platform that replaces manual integration of devices with a scalable, plug-and-play mechanism that treats both sensors and actuators as first-class entities. The platform should enable spaces that can be assembled programmatically by software developers instead of hardwired by engineers and system integrators.

Atlas is a new, commercially available sensor and actuator platform that enables the concepts of self-integrative, programmable pervasive spaces [3, 4]. It provides physical nodes for connecting numerous heterogeneous devices, a mechanism for translating those devices into software services, a system for maintaining a library of device services and their interfaces, and a runtime environment for accessing services and composing applications. Atlas is designed specifically to be the basic building block for programmable pervasive spaces.

This paper will provide a brief overview of the various components of the Atlas platform followed by a detailed guide on how to use the platform. We then present three major Atlas-based case studies from the Gator Tech Smart House (GTSH) [3]: the Smart Blinds, the Smart Floor, and the Smart Front Door. By focusing on the tasks necessary to use the new platform, these examples serve as a guide for other researchers wishing to use Atlas. We conclude with a brief section describing both the ongoing work of migrating the remaining GTSH applications to Atlas and the upcoming features in the next release of the platform itself.

2. The Atlas Platform

The Atlas Platform [5] is a combination of hardware nodes, firmware running on the hardware, and a software middleware that provides services and an execution environment. Together, these components allow virtually any kind of sensor, actuator or other device to be integrated into a network of devices, all of which can be queried of controlled through an interface specific to that device, and facilitates the development of applications that use the devices. This section will provide a brief overview of the hardware and middleware components of the platform.

Each Atlas node is a modular hardware device composed of stackable, swappable layers, with each layer providing specific functionality. The basic Atlas node configuration (Fig. 1, left) consists of three layers, the Processing Layer, the Communication Layer and the Device Connection Layer. The modular design and easy, reliable quick-connect system allow users to change node configurations on the fly. This feature proved extremely useful for us since in order to connect a myriad of different devices in the house we required different device connection layers, whose details are provided in the following sections. The rest of the hardware configuration essentially remained the same for all the nodes deployed in the house, consisting of the Processing Layer based on the Atmel ATmega128L microcontroller and the Communication Layer based on the Cirrus Logic Crystal LAN CS8900 NIC IC which provides basic 10 Base-T wired Ethernet networking.

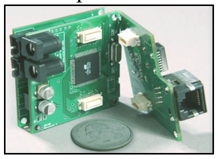
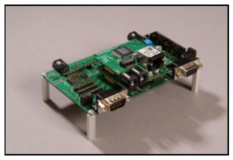



Figure 1. The Atlas platform (left); programming an Atlas node (right).

The Atlas platform uses OSGi [6] as the basis of its middleware. OSGi provides the many service discovery and configuration mechanisms necessary to create programmable pervasive spaces. The Atlas middleware consists of the SensorNetworkManager bundle that listens for connections from Atlas nodes, manages the sensor and actuator service bundles uploaded by the nodes, and provides mechanisms for connecting these bundles with other services and applications running in the framework.

3. Preparing the Atlas Platform

3.1. Obtaining the platform

The Atlas platform can be obtained from http://www.sensorplatform.org. The number of nodes to order is based on several factors, such as number of sensors and actuators, power requirements of actuated devices, and the distance between devices. For example, 32 analog sensors in the same geographical area could be connected to a single node, but 2 sensors at opposite ends of a smart house would likely be easier to integrate using two different nodes.

Each node requires some Communication Layer to be selected. Currently wired Ethernet and WiFi are available, and a ZigBee Layer is under development. SensorPlatform.org also offers a variety of Device Connection Layers to complete the nodes. The current selection consists of the 8-way analog sensor board, 32-way analog sensor board, 6-way servo board and the 6-way motor driver board. The sensor and servo boards are compatible with Phidgets devices.

3.2. Atlas out of the box

The Atlas package will include the ordered Processing Layers, Communication and Device Connection Layers. The layers are swappable (each Processing Layer can connect with any Communication or Device Connection Layer).

The package contains a power supply for each node, and a software distribution CD containing the software required to use the Atlas platform which consists of:

a) The Atlas middleware: a single OSGi bundle, SensorNetworkManager.jar.
b) Selection of available sensor, actuator service bundles from the Atlas library (including source).
c) The Atlas node firmware source code.
d) Development tools: JDK 1.4.2, avr-gcc (with Programmer's Notepad IDE), AVR Studio 4, Knopflerfish OSGi [7].

Cables such as Serial and Ethernet are not included. Additionally, node-programming hardware such as ISPs are not included, although they are currently necessary if users need to configure a node or modify the Atlas firmware. Work is underway to develop a web interface (Fig. 2, right) for configuring the Atlas nodes directly over the network. For basic node configuration operations, this will eliminate the need to modify the firmware and purchase additional node-programming hardware.

3.3. Installing the Atlas middleware

To simplify deployment in spaces such as smart homes, the Atlas middleware is able to run on any standard PC. Because the middleware is based on OSGi, a Java-based framework, it can run on any operating system that has a JVM. Many implementations of the OSGi specification exist. The open source Knopflerfish implementation of the OSGi framework is provided in the software distribution

CD. Once the OSGi framework is installed, users then load and activate the Atlas middleware bundle. To install the middleware the following steps need to be taken:

a) Install JDK 1.4.2 (if one wants to develop additional Atlas service bundles).
b) Install the Knopflerfish OSGi framework provided in the software distribution CD by running the self extracting jar file.
c) Once Knopflerfish is installed, load it by running 'java –jar framework.jar'.
d) Click on 'Open Bundles' and select SensorNetworkManager.jar from the CD. This installs and starts up the Atlas middleware.

3.4. Testing nodes with sample applications

The Atlas nodes come pre-programmed with demo software to allow new users to get acquainted with their operation. Connect any standalone computer and the Atlas nodes to an Ethernet hub. Set the IP address of the computer to 192.168.1.201. Load the Knopflerfish OSGi framework and the Atlas middleware on this computer and then power up the nodes. After a delay of a few seconds, all the nodes which have been powered up will automatically appear as active services in the OSGi framework. On selecting any of the services shown in the framework, the user can get details about the service and the methods provided to access that service. If any of the nodes are powered off, the services provided by them will disappear from the framework.

3.5. Configuring the Atlas nodes

Currently, users must modify the Atlas firmware to configure the nodes for their particular network. This firmware controls all the sensors and actuators connected to the node. It also manages the communication between the node and the Atlas middleware. Additionally, it uploads the service bundle (stored onboard the node) to the OSGi framework, where the bundle is registered as a service representing all the devices connected to the node. To modify the firmware, users must perform the following steps:

a) Install avr-gcc (includes Programmer's Notepad) and AVR Studio from CD
b) Start Programmer's Notepad and load the SensorPlatformOS project file, which contains the source code for the Atlas firmware.
c) From the list of files that are loaded, open the Atlas node configuration file, *sensornode_conf.h*.
d) In this file, the following property values need to be modified (Table 1 provides the configuration of a typical node):
    (i) NODEID: Unique ID of node
    (ii) SP_MAX_ANALOG_SENSORS: Number of analog sensors connected
    (iii) EEPROM_SIZE: Size (in bytes) of OSGi service bundle stored on node
    (iv) SPM_IPADDR0 – SPM_IPADDR3: IP address of machine hosting OSGi
    (v) UIP_IPADDR0 – UIP_IPADDR3: IP address of node
    (vi) UIP_DRIPADDR0 – UIP_DRIPADDR3: IP address of default router
e) In Programmer's Notepad select 'Tools -> Make All' to compile the firmware. The output will a file called *nodeFirmware.hex*.

Table 1. Typical node configuration values

Property	Value	Property	Value
NODEID	20	UIP_IPADDR0	192
SP_MAX_ANALOG_SENSORS	32	UIP_IPADDR1	168
EEPROM_SIZE	4013	UIP_IPADDR2	1
SPM_IPADDR0	192	UIP_IPADDR3	220
SPM_IPADDR1	168	UIP_DRIPADDR0	192
SPM_IPADDR2	1	UIP_DRIPADDR1	168
SPM_IPADDR3	201	UIP_DRIPADDR2	1
		UIP_DRIPADDR3	1

This same procedure should be followed by users wishing to modify the firmware to support new features.

3.6. Preparing the Atlas service bundle

The service bundle stored onboard a node defines the type of devices connected to that node. This definition includes the services and methods available for other applications to access and control the different devices. Fig. 2 (left) shows the services and methods exported by one of the bundles running at the GTSH.

These bundles are uploaded and registered on the OSGi framework whenever an Atlas node powers up. This means that whenever a new component or device is introduced to the house and powered on, it automatically appears as an OSGi service. This service can be used immediately by the different smart home applications, thereby making it plug-and-play. For example, if a new window blind is added to the house and powered on, it can be accessed and controlled by the existing blinds control application without requiring any re-configuration or re-compiling of the existing software.

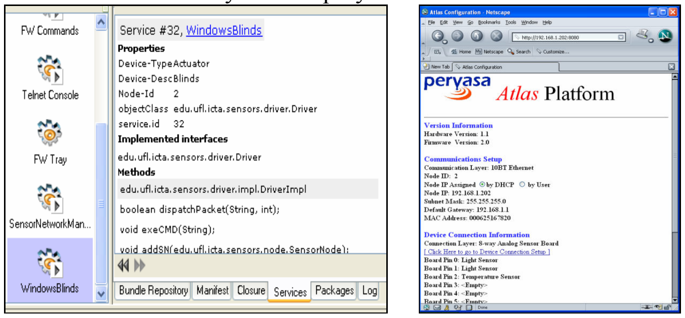
The software distribution CD provides generic, open source service bundles for analog sensors and actuators. The user may either use these bundles directly or create their own bundles by implementing the necessary Java interfaces. All Atlas service bundles implement the 'Driver' interface, details of which are included in the support documentation. The various jar files which need to be included by users if they choose to write their own bundle are also included, namely, SensorNetworkManager.jar and framework.jar.

SensorNetworkManager.jar and framework.jar (which need to be included by users if they choose to write their own bundles) are also included. The source code for all the pre-built service bundles is provided under the GPL agreement.

3.7. Uploading the Atlas firmware and service bundle

Upload the Atlas firmware and service bundle using the following process:

a) Mount Processing Layer on debug board.
b) Attach AVRISP to debug board (Fig. 1, right) and to development machine using a serial port.
c) Power on Atlas node, then load AVR Studio.
d) Select 'Tools->Program AVR->Connect' and from the list, select AVRISP and the serial port it is connected to. Click 'Connect'.
e) After AVR Studio connects to node, select nodeFirmware.hex as the file to be programmed into the FLASH and the service bundle jar file as the file to be stored in the EEPROM.
f) Click on 'Program FLASH' and after it finishes programming, click on Program EEPROM.
g) The Atlas node is now ready to be deployed.



```
Service #32, WindowsBlinds
Properties
Device-TypeActuator
Device-DescBlinds
Node-Id     2
objectClass edu.ufl.icta.sensors.driver.Driver
service.id    32
Implemented interfaces
edu.ufl.icta.sensors.driver.Driver
Methods
edu.ufl.icta.sensors.driver.impl.DriverImpl
boolean dispatchPacket(String, int);
void exeCMD(String);
void addSN(edu.ufl.icta.sensors.node.SensorNode);
```

http://192.168.1.202:8080

Version Information
Hardware Version: 1.1
Firmware Version: 2.0

Communications Setup
Communication Layer: 10BT Ethernet
Node ID: 2
Node IP Assigned [✓ by DHCP] by User
Node IP: 192.168.1.202
Subnet Mask: 255.255.255.0
Default Gateway: 192.168.1.1
MAC Address: 000625167820

Device Connection Information
Connection Layer: 8-way Analog Sensor Board
[Click Here to go to Device Connection Setup]
Board Pin 0: Light Sensor
Board Pin 1: Light Sensor
Board Pin 2: Temperature Sensor
Board Pin 3: <Empty>
Board Pin 4: <Empty>
Board Pin 5: <Empty>

Figure 2. Services and methods offered by WindowsBlinds service bundle (left); Web Interface (right).

4. Case Studies

The Gator Tech Smart House (Fig. 3, left) is a 2,500 sq. ft. pervasive computing environment located in the Oak Hammock retirement community in Gainesville, Florida. Opened January 28, 2005, the GTSH showcases many technologies and services designed to assist both elderly residents and local or remote caregivers.

Since the Atlas platform was not available until late 2005, the original implementation of the GTSH used other sensor platforms (such as Phidgets) and automation technologies (such as X10 modules). In preparation for the first in a series of experiments with live-in residents, we currently are migrating the existing services and applications inside the GTSH to the Atlas platform. Additionally, we are creating new services made possible by the new platform. The following case studies provide details of our experience.

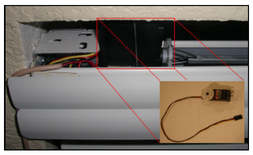


Figure 3. The Gator Tech Smart House (left); Smart Blinds control mechanism (right).

4.1. Smart Blinds

One of the major aims of the GTSH is to allow its resident to control various household devices, such as the window blinds, using voice commands or a simple touch screen, interactive GUI (Fig. 5, right). Our plan was to deploy a system which would not only allow the resident to operate the blinds without requiring physical interaction but also allow the house to control them automatically to adjust ambient lighting.

Each window shade in the house is connected to a Hi-Tec HS-322HD Deluxe Servo (Fig. 3, right) with an output torque of 3kg.cm, which allows the smart house to open and close the blinds. The servos are connected to Atlas nodes via the 6-way servo board, allowing a single node to control up to 6 servos.

4.1.1. The Window Blinds service

The WindowBlinds service was implemented using the generic actuator service bundle. This service bundle translates the high-level commands provided by the end-user application into low level instructions required by the Atlas node to control the servos. It allows an application to both control individual blinds and also control multiple blinds as a single entity.

4.2. Atlas-based Smart Floor

In a smart home geared towards providing an assistive environment for seniors, locating the residents and keeping track of their whereabouts is of paramount importance. Indoor location tracking systems provide information about the resident's location, daily activities, and room preferences and also help in detecting emergencies like falls. In addition to this, it is equally essential that such a system should neither be intrusive and nor should it require special attention from the resident to operate effectively. Keeping these things in mind it was decided to install a Smart Floor in the GTSH which would provide unencumbered indoor location tracking using pressure sensors located beneath floor tiles. Kaddoura et al. [8] describe such a system where a pressure sensor is centrally placed underneath each square foot block of the floor and is able to detect a foot step on any part of that block. This system is not only able to provide nearly 100% coverage over its area of deployment but is also relatively inexpensive as compared to other similar location tracking systems in use today.

The Gator Tech Smart House has a residential-grade raised floor consisting of floor tiles measuring one square foot each. The process of deploying the piezoelectric pressure sensors was the same as described in [8].

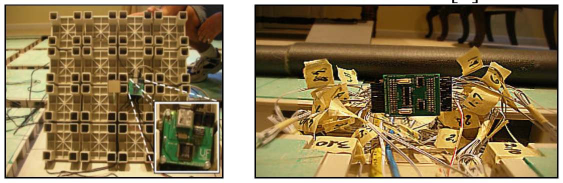


Figure 4. A Smart Floor tile with Atlas node (left); 32-way analog sensor board (right).

4.2.1. Connecting the Atlas nodes

The approach taken by Kaddoura et al. had the pressure sensors connected to Phidgets 8/8/8 Interface Kits, which can only support a maximum of 8 sensors. For the second iteration of the Smart Floor, we used the Atlas platform (Fig. 4, left) together with its 32-way analog sensor board (Fig. 4, right), which supports 32 two-wire analog sensors. In this manner we were able to deploy the Smart Floor throughout a large section of the house (over 2000 sq. ft.) using only ten Atlas nodes. This improved the cost-effectiveness of the system. We also made use of the on-board filtering capability of the platform to only transmit sensor data if there is a change in readings beyond a user-defined threshold. This prevents the Smart Floor from flooding the entire sensor network in the house.

4.2.2. The Smart Floor service

The Smart Floor service was implemented using the generic analog sensor service bundle mentioned in Sec. 3.6. Applications, such as the Location Tracker (Fig. 5, left), access the Smart Floor sensor readings simply by subscribing as a listener to the dispatchPacket event produced by the bundle.

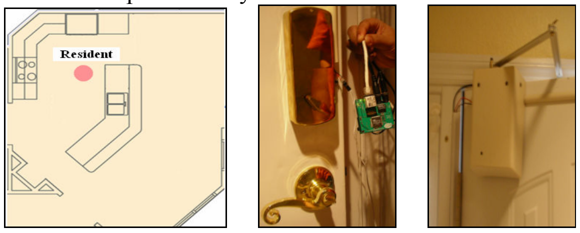


Figure 5. Location Tracker (left); Deadbolt mechanism (center); Door opener (right)

4.3. Smart Front Door

We created an intelligent front door in the GTSH to support elderly and disabled residents. The door facilitates access to the house by means of a keyless entry system using RFID badges and an automatic door opening, closing and locking mechanism. Voice control of the door also allows the resident to grant access to visitors without having to move to the foyer.

The door is fitted with a Mi-KF01P deadbolt locking mechanism (Fig. 5, center) offering both electronic operation for keyless entry and conventional operation using keys. The deadbolt turning mechanism is driven by a standard 5V DC motor connected to an Atlas node using a motor driver board. For opening and closing the door we use the Private-Door Duo door opener (Fig. 5, right) because it allows both automatic and manual operation.

4.3.1. The Front Door service

The Front Door service bundle coordinates the functioning of the door lock and the door-opening mechanisms. It provides services to the application developers to both lock/unlock the door and also to open/close it if necessary. It also makes sure that conflicting commands issued to the Atlas node controlling the door are not executed, for example, opening a door which is locked. The lock/unlock commands issued by this bundle are executed by the Atlas node controlling the locking mechanism while the door open/close commands are issued as X10 commands to the door opener.

Using this service bundle, one of the co-authors developed an application which allows residents keyless entry into the Smart House via the use of RFID badges and also allows the resident to operate the front door by issuing simple voice commands from anywhere inside the house.

5. Work in Progress

5.1. Atlas Platform

A number of new features are currently under development. The most important of these is a web interface to configure the Atlas nodes without requiring the users to modify the firmware (Fig. 2, right). Work is also underway to develop a bootloader for Atlas that allows new firmware to be loaded over the network or via serial port without the use of node-programming hardware. As for the node hardware itself, we plan to add a wide range of new communication layers such as ZigBee, Bluetooth and power line communication.

5.2. Gator Tech Smart House

Live-in trials began in the GTSH on March 24th 2006. The subjects' activities are being monitored and logged for analysis by our collaborators in the Department of Occupational Therapy. Writing the logging application was a straight-forward task because Atlas provides a homogeneous interface to the plethora of hardware devices installed in the house. Like other GTSH applications, the logger is an OSGi bundle running in the framework. It can subscribe to events from various service bundles and from other applications to record data produced in the house.

Currently we are also working on replacing the X10-controlled lights, the SmartWave microwave [9], and the SmartPlugs [10] in the GTSH.

References

[1] http://www.xbow.com/Products/Wireless_Sensor_Networks.htm

[2] S. Greenberg and C. Fitchett, "Phidgets: easy development of physical interfaces through physical widgets," Proc. of 14th ACM Symp. on User Interface Software and Technology, Nov. 2001.

[3] A. Helal et al.,"Gator Tech Smart House: A programmable pervasive space" in IEEE Computer,vol.38,no.3, pp.50-60, Mar.2005.

[4] S. Helal, "Programming pervasive spaces," in IEEE Pervasive Computing, vol. 4, no. 1, 2005.

[5] http://www.sensorplatform.org

[6] D. Maples and P. Kriends, "The Open Services Gateway Initiative: An introductory overview," in IEEE Comm. Magazine, vol. 39, no. 12, pp. 110-114, 2001.

[7] http://www.knopflerfish.org

[8] 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.

[9] J. Russo, A. Sukojo, S. Helal, R. Davenport and W. Mann, "SmartWave intelligent meal preparation system to help older people live independently," Proc. of the 2nd. Intl. Conf. on Smart homes and health Telemetrics, pp. 122-135, Sept. 2004.

[10] H. El-Zabadani, A. Helal, B. Abdulrazak and E. Jansen, "Self-sensing spaces: smart plugs for smart environments," Proc. of 3rd Intl. Conf. on Smart homes and health Telemetrics, July 2005.