Atlas: A Service-Oriented Sensor Platform

Hardware and Middleware to Enable Programmable Pervasive Spaces

Jeffrey King, Raja Bose, Hen-I Yang, Steven Pickles, Abdelsalam Helal
Mobile & Pervasive Computing Laboratory, CISE Department
University of Florida, Gainesville, FL 32611, USA, http://www.icta.ufl.edu
{jck, rbose, hyang, spickles, helal}@cise.ufl.edu

Abstract

Pervasive computing environments such as smart spaces require a mechanism to easily integrate, manage and use numerous, heterogeneous sensors and actuators into the system. However, available sensor network platforms are inadequate for this task. The goals are requirements for a smart space are very different from the typical sensor network application. Specifically, we found that the manual integration of devices must be replaced by a scalable, plug-and-play mechanism. The space should be assembled programmatically by software developers, not hardwired by engineers and system integrators. This allows for cost-effective development, enables extensibility, and simplifies change management. We found that in a smart space, computation and power are readily available and connectivity is stable and rarely ad-hoc. Our deployment of a smart house (an assistive environment for seniors) guided us to designing Atlas, a new, commercially available service-oriented sensor and actuator platform that enables self-integrative, programmable pervasive spaces. We present the design and implementation of the Atlas hardware and middleware components, its salient characteristics, and several case studies of projects using Atlas.

1. Introduction

The UF Mobile and Pervasive Computing Lab is dedicated to applied research on assistive environments for seniors and the disabled. Our largest project is the Gator Tech Smart House [1] (GTSH), a 2500 sq. ft. pervasive computing environment located in the Oak Hammock retirement community in Gainesville, Florida. The GTSH showcases many technologies and services designed to assist both elderly residents and local or remote caregivers.

While using the GTSH as a showcase is useful, its main benefit is as a test-bed for the fundamental components and practices that turn a normal space into a pervasive computing smart space.

Most first-generation pervasive space prototypes in existence now are the result of massive ad-hoc system integration. Introducing a new device to the environment is a laborious process. After the initial decision over which particular component to purchase, the smart space developers must research the device's characteristics and operation, determining how to configure it and interface with it. The device must then somehow be connected and physically integrated into the space. Any applications using the new device must be written with knowledge of the resources assigned to connect the device, signals to query and control the device, and the meaning of any signals returned. Finally, tedious and repeated testing is required to guard against errors or indeterminate behavior that could occur if, for example, applications make conflicting requests of devices, or if devices or connection resources themselves conflict. Any change in deployed devices or applications requires repeating the process. This is the problem with integrated pervasive spaces.

The goal of the Pervasive Computing Lab at the University of Florida is to develop models, methodologies, and processes for creating programmable pervasive spaces [2]. This is a concept in which a smart space exists, in addition to its physical entity, as a runtime environment and a software library [3]. Service discovery and gateway protocols and frameworks (such as OSGi [4,5]) automatically integrate system components using a generic middleware that maintains a service definition for each sensor and actuator in the space. Programmers assemble services into composite applications using various programming models [6], tools, and features of the middleware.

The Atlas sensor platform described in this paper is the basic building block for programmable pervasive spaces. Atlas provides physical nodes for connecting various heterogeneous devices, a system 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.

The remainder of this paper is organized as follows: Section 2 examines related sensor network research and indicates why these platforms are not ideal for developing pervasive computing spaces. Section 3 analyzes the role of sensors and actuators in pervasive computing spaces, analyzes how these devices can be integrated into a system in a way that facilitates the development of the pervasive space and the applications and services that run in the space. Section 3 also presents our overall architecture for programmable pervasive spaces including the requirements for a sensor and actuator platform that enables these spaces. Section 4 details the implementation of Atlas, a service-oriented sensor and actuator platform that enables the creation of programmable pervasive spaces. Section 5 illustrates the deployment of an Atlas-based system, from configuring nodes to building applications using our Eclipse plug-in. Section 6 references case studies from various projects (both within our lab and in collaboration with other groups) using the Atlas platform. Section 7 provides a comparison matrix describing features and characteristics of Atlas and other sensor platforms. Section 8 concludes the paper with a summary of the current status of the Atlas platform and information about upcoming releases.

2. Related Work

There has been a dramatic increase during the past three years in the number of sensor platforms in development or commercially available. The most visible of these has been the Mote family, developed by the University of California at Berkeley as part of the Smart Dust [7] project. Motes such as the MICAz, MICA2, and MICA2DOT are available commercially from Crossbow Technologies [10]. These platforms include an integrated processing and communication module and offer limited modularity in the form of daughter cards, containing different sensor arrays, which can be plugged into the platform. Other versions lack this modularity. For example, Telos [9], as developed by the Smart Dust team, is a completely integrated platform based on the TI MSP430 microcontroller. It offers higher performance and consumes less power than other Mote platforms, but comes at a higher cost, and the available sensors are integrated into the device and cannot be changed by users.

Many groups are working with Motes either as the basis for other projects or to further the sensor platform itself. Intel and Berkeley have worked together on iMote [10], a Bluetooth-enabled version of the wireless sensor node. College of the Atlantic collaborated with Berkeley to use wireless sensor networks for habitat monitoring on Great Duck Island [11].

Motes are currently the de facto standard platform for sensor networks. Although the Mote was primarily developed for use in wireless ad-hoc networks for applications such as remote monitoring, researchers in many unrelated areas have used Mote primarily for its commercial availability and its ability to integrate numerous sensors into a system.

Phidgets [12], developed by the University of Calgary, is another widely used, commercially available platform. The Phidgets support a large variety of sensors and actuators. They allow rapid application development and are extremely easy to use. But the Phidgets are not fully modular, and they only support communication to a Windows desktop computer via USB, which leads to scalability problems.

Some groups have worked on creating a more modular sensor network platform. The Cube [13], developed by University College Cork, and MASS [14], a Sandia National Laboratory project, have modular architectures allowing users to rapidly develop applications and reconfigure platforms as necessary. Other sensor network platforms, such as NIMS [15], XYZ [16], and Eco [17] were designed for specific applications: environmental monitoring (NIMS, XYZ) and health monitoring (Eco).

The Smart-Its [18], developed jointly by Lancaster University and the University of Karlsruhe, offer some features that could facilitate the development of pervasive spaces. They have a somewhat modular hardware design and a template-based software design process, which allows rapid application development. But the Smart-Its platform is still not completely modular, with an integrated processing and communication board. Furthermore, devices connected through Smart-Its are constrained to a single application (running on the Smart-It hardware). This does not allow for service-rich environments in which applications can be developed using service composition.

None of the available sensor network platforms are fully adequate for the development of pervasive spaces. Most of the platforms focus only on sensors, and barely touch upon the issue of actuators. In a pervasive space, actuators play as important a role as sensors, as actuators are used to influence the space. NIMS and XYZ make use of actuators, but only for the specific purpose of making the platforms mobile. Phidgets support a large number of actuators, but are constrained by scalability issues and a fixed hardware configuration.

Additionally, none of these 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. Except for the larger number of devices supported, this is no better than connecting sensors and actuators directly to the I/O ports of a computer. It is a development method that does not scale as more devices and services are added to a smart space.

These shortcomings lead us to create a modular, service-oriented sensor and actuator platform specifically designed to support the development of pervasive computing spaces.

3. Sensors, Actuators for Pervasive Spaces

Our lab has thoroughly investigated [2,3,4,6,19,20] how to develop robust, maintainable, and service-rich pervasive spaces. During this research, we presented a formal model of pervasive spaces, which first required identifying the key entities in such a space. To summarize, a space consists of living beings and objects. The living beings interact with each other and with the objects. In a pervasive space, the living beings are users, and we can divide the objects into two categories: passive objects and active objects.

Passive objects are "dumb" objects that cannot be queried or controlled by the smart space. At best, passive objects may be recognized by the space, but only users can manipulate them. Passive objects therefore are not key entities in a smart space. Active objects, however, can provide information to, or be manipulated by, the smart space. Active objects are key entities.

Active objects are further divided into two classes: sensors and actuators. Sensors provide information about a particular domain, supplying data to the system about the current state of the space. Sensors only provide measurement; they cannot directly alter the state of the space. Actuators are the active objects that alter the space. They activate devices that perform certain functions.

Sensors and actuators are the foundation of a pervasive space, as they provide the means for gathering information about the state of the space and for controlling devices that can modify the state of the space. We therefore require a platform to connect numerous and heterogeneous sensors and actuators to the services and applications that will monitor and control the space.

Connecting sensors and actuators to applications implies more than simply physically coupling these devices to a computer platform (although this is certainly important, as we wish to employ far more devices than could be connected to the limited I/O ports for a single machine or even a small cluster). Connecting devices with applications means providing some mechanism for the applications to make use of devices and services directly, instead of accessing some I/O resource on a machine that happens to be wired to a particular device. Beyond dealing with resource allocations, connecting applications and devices means eliminating the need for those applications to know the low-level information (voltages, control codes, etc.) to drive the devices.

To solve this problem, we require a network-enabled, service-oriented platform that can "convert" the various sensors and actuators to software services. The required sensor platform would be responsible for obtaining this representation and for managing the services in such a way that applications are easily able to obtain and use the services and associated knowledge. Realizing this, we were able to design the architecture for programmable pervasive spaces shown in Fig. 1.

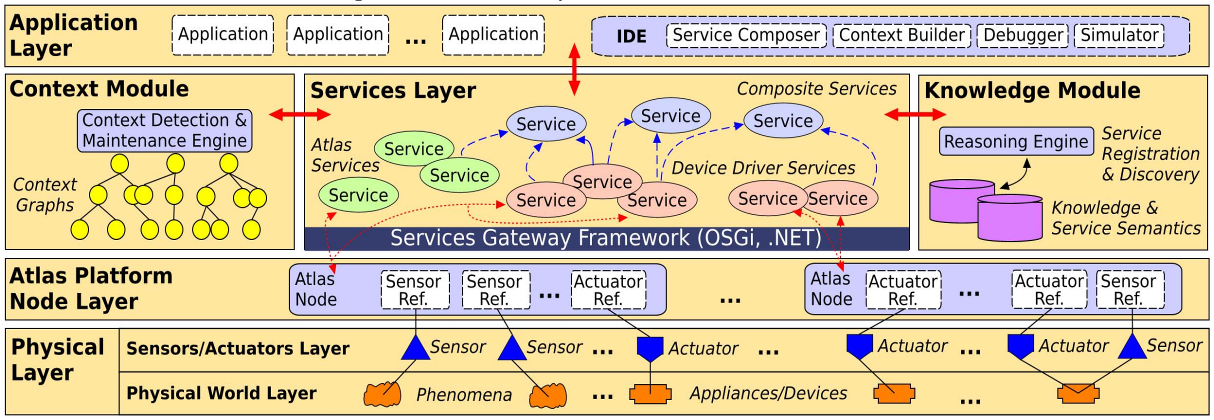


Figure 1. Middleware architecutre for programmable pervasive spaces.

The physical layer contains passive and active objects. Through sensors and actuators, active objects are captured into the smart space for observation and control.

The platform node layer, implemented by Atlas, contains all the sensor and actuator platform nodes in the environment. These nodes automatically integrate the sensors and actuators (and hence their respective active objects) from the layer beneath and export their service representations to the layers above.

The service layer, which resides above the platform layer, holds the registry of the software service representation of all sensors and actuators connected to the platform nodes. The service layer, which typically runs on a centralized, full-fledged server, also contains the service discovery, composition, and invocation mechanisms for applications to locate and make use of particular sensors or actuators. The service layer contains a context-management extension as well as a knowledge representation and storage extension, both of which are necessary to build programmable pervasive spaces.

Finally, the application layer sits at the top and consists of the execution environment that provides access to the software library of sensors, actuators, and other services. It also contains the actual applications and composed services that monitor and control elements of the pervasive space.

Following this architecture, we created the Atlas sensor and actuator platform to address the inadequacies of existing platforms for creating programmable pervasive spaces. The Atlas platform covers the Sensors/Actuators, Node, and Services Layers in Fig. 1.

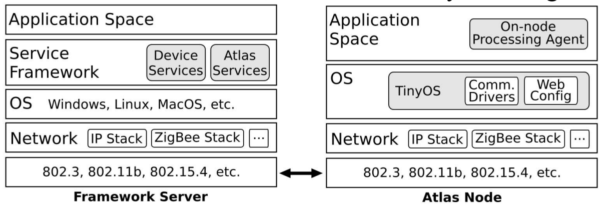


Figure 2. Software architecture of the Atlas platform.

Fig. 2 shows the layered software architecture of the Atlas node and the full-fledged server that hosts the service framework. The Atlas driver runs on the Atlas node. On power-up, it registers the associated sensor or actuator services on the framework server. Optionally, a processing agent could be dynamically loaded onto the Atlas node to allow for on-node processing (such as data filtering).

4. The Atlas Platform

The Atlas platform is a combination of hardware, 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 or controlled through an interface specific to that device, and facilitates the development of applications that use the devices. This section will describe in detail the implementation of each element of the platform.

4.1. Hardware

Each Atlas node is a modular hardware device composed of stackable, swappable layers, with each layer providing specific functionality. The modular design and easy, reliable quick-connect system allow users to change node configurations on the fly.

A basic Atlas node configuration (Fig. 3) consists of three layers: the Processing Layer, the Communication Layer, and the Device Connection Layer.

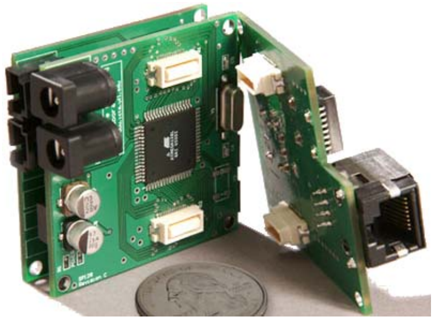


Figure 3. A three-layered Atlas node.

4.1.1. Processing Layer. The Processing Layer is responsible for the main operation of the Atlas node. Our design is based around the Atmel ATmega128L microcontroller. The ATmega128L is an 8MHz chip that includes 128KB Flash memory, 4KB SRAM, 4KB EEPROM, and an 8-channel 10-bit A/D-converter. The microcontroller can operate at a core voltage between 2.7 and 5.5V. We chose this chip for its low power consumption, plethora of I/O pins, ample SRAM and program space, and the readily available tools and information resources. In addition to the ATmega128L, we include 64KB of expanded RAM for on-node services and a real-time clock for accurate timing. This clock can also be used to have the microcontroller wake from a sleep state at specified intervals.

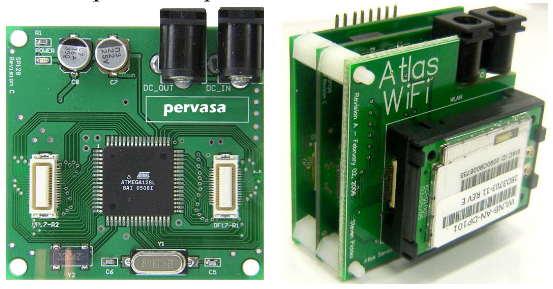


Figure 4. Processing Layer (left), Atlas Node with WiFi Communication (right).

Whereas most current commercially available platforms are concerned only with sensors, Atlas treats actuators as a first-class entity, and the power supply design on the Processing Layer reflects this design: the platform supports both battery power and wired power.

Battery power is an obvious choice in sensor networks, where applications often require long-lived, unattended, low-duty cycle sensing. The other option, wired power, may seem unusual in this field. But given the primary goal of Atlas – enabling the development of smart spaces – wired power is a necessity. Smart spaces will contain many actuators, and driving these devices requires much more power than operating the platform nodes. Hence, Atlas supports both wired and battery power.

In case wired power is used, the second plug can be used to daisy-chain nodes together, reducing the number of outlets used in a smart house environment. The number of nodes that can be chained to a single power supply depends on the number and type of devices connected to the platforms. For example, in the Gator Tech Smart House Smart Floor (see Sec. 6.1), each node is connected to 32 pressure sensors and 15 Atlas nodes can be daisy chained easily.

4.1.2. Communication Layer. Data transfer over the network is handled by the Communication Layer. Currently we have four options – wired 10BaseT Ethernet, 802.11b WiFi, Bluetooth, and USB, with the first two being the most popular.

The wired Ethernet Communication Layer shown in Fig. 4 uses a Cirrus Logic Crystal LAN CS8900a NIC IC and a standard RJ45 connector for basic 10 Base-T networking. LEDs provide instant feedback as to power, connectivity, and LAN activity. Wired Ethernet is important in situations requiring high-speed data access over an extremely reliable connection. For example, the Gator Tech Smart House uses wired Ethernet Atlas nodes for critical systems such as location/fall detection. This is also the ideal layer for applications where nodes are situated in areas shielded from RF communication.

The WiFi Communication Layer shown in Fig. 4 is based on the DPAC WLNB-AN-DP101 Airborne Wireless LAN Module, providing 802.11b connectivity to the Atlas Platform. It allows Atlas nodes to be deployed easily, without the hassle of laying out cables, when the reliability of constant connectivity is unneeded.

In addition to the above communication options, a ZigBee layer, based on the Chipcon CC2420 transceiver chip, is being finalized. Further details are provided in Sec. 8.1.

4.1.3. Device Connection Layer. The Connection Layer is used to connect the various sensors and actuators to the platform. Integrating any number of analog and digital sensors is simple and easy. We currently have layers capable of connecting up to 32 sensors to a single node.

We also provide connection layers for actuators and have produced boards capable of controlling servos, motors, and LEDs. We also have a TRIAC layer that lets Atlas control the operation of household electrical devices (without disabling the standard manual control), allowing our platform to replace X10 home automation modules and similar devices.

IEEE 1451 [21] defines a standard for sensor or actuator (or any transducer) connections, including support for providing a transducer electronic data sheet (TEDS) at the connection level. We intend to support this standard in the next release of the Atlas platform.

4.1.4. Other Layers. Atlas is not limited to three layers. Additional layers could be added to provide extra processing power, security features, multiple communication mediums, network switching, or alternative power options.

4.2. Firmware

The firmware runs on the Processing Layer of the Atlas platform hardware, and allows the various sensors, actuators, and the platform itself to automatically integrate into the middleware framework.

The firmware first detects the type of Communication Layer attached to the Atlas node. It then connects to the Atlas middleware and uploads its configuration details, which registers all the devices connected to that particular node, making the devices available as OSGi services in the framework.

After this initialization process concludes, the node goes into data-processing mode. It begins sending data from sensors and receiving commands to control sensor and actuator operations.

4.3. Middleware

Although the middleware does, in part, run on the Atlas nodes, the majority of the framework operates on a stand-alone PC. We currently use OSGi as the basis of our middleware. OSGi provides many service discovery and configuration mechanisms we need to create programmable pervasive spaces.

As mentioned previously, when an Atlas node comes online, it performs some negotiation with the middleware. Specifically, it is connecting to the Network Listener, which is listening on a dedicated port for new nodes joining the network.

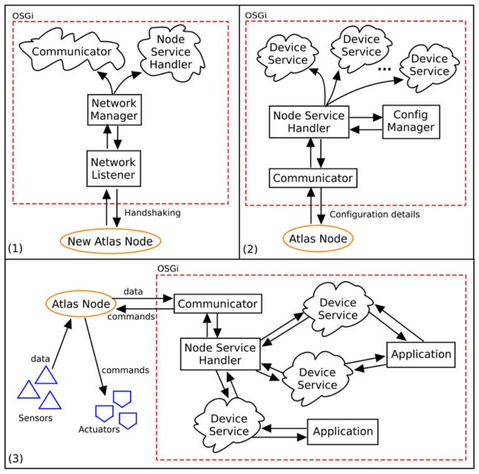


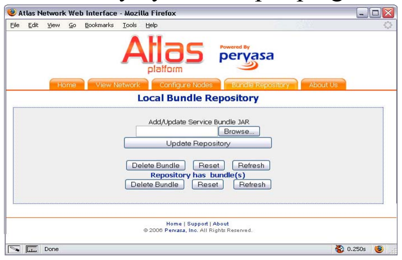
Figure 5. Atlas middleware framework.

As shown in Fig. 5(1), after the initial handshake, the Network Manager spawns a Communicator Thread that will exclusively handle all the network communications with this particular node from now on. A Node Service Handler (NSH) is also created which registers the various devices connected to this node as OSGi services (as shown in Fig. 5(2) and handles the routing of commands and data between the service bundles and their respective devices. Applications are then able to locate and use these services provided by the new devices (Fig. 5(3)).

5. Deployment & Application Development

5.1. Configuring the Nodes

Most other sensor platforms require the modification, re-compilation and re-deployment of firmware every time node settings are modified or different sensors or actuators are connected to the platform. Atlas, however, provides an intuitive, easy-to-use web interface (Fig. 6, 7) for configuring the node and selecting the sensors and actuators to be connected to it. After saving the new configuration, the node automatically restarts itself and the new devices show up in the framework as OSGi services. Programmers can then develop applications that use these services -- a task that is made easy by our Eclipse plug-in.



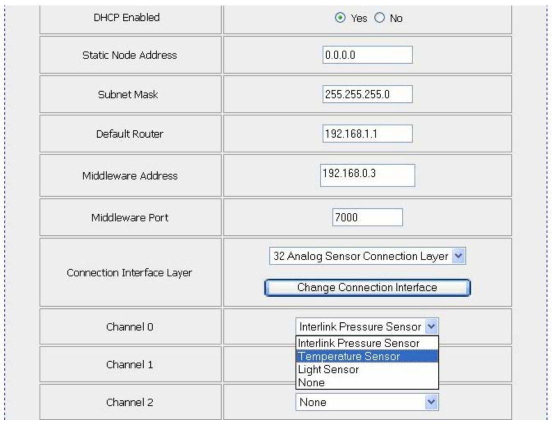
Local Bundle Repository

Add/Update Service Bundle JAR

Repository has bundle(s)

© 2006 Pervasa, Inc. All Rights Reserved.

Figure 6. Atlas web configuration for bundle repository.

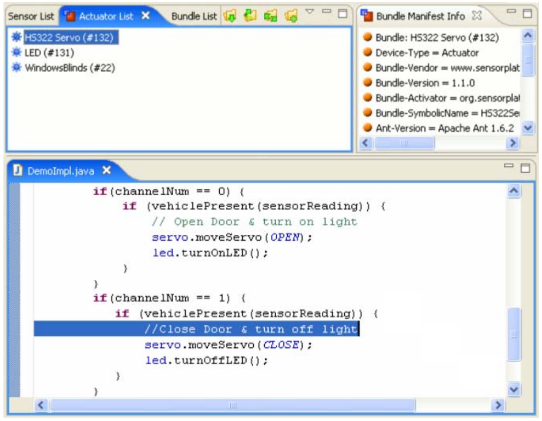


DHCP Enabled	Yes [✓ No]
Static Node Address	0.0.0.0
Subnet Mask	255.255.255.0
Default Router	192.168.1.1
Middleware Address	192.168.0.3
Middleware Port	7000
Connection Interface Layer	32 Analog Sensor Connection Layer
Channel 0	Interlink Pressure Sensor
Channel 1	
Channel 2	None

Figure 7. Atlas web configuration for nodes.

5.2. Eclipse IDE Plug-in

We have implemented an Atlas plug-in for Eclipse (Fig. 8) as part of the effort to make pervasive computing environments programmable. The IDE improves programmability by providing assistance in several aspects. It offers the capability to connect to remote Atlas frameworks, retrieving the list of available sensors and actuators, as well as existing application bundles. It also obtains information about the bundles, such as dependency relationships, packaging, and versioning. The plug-in takes a snapshot of the remote runtime environment (which can be dynamically refreshed), allowing programmers to work without a constant connection to the space.



HS322 Servo (#132)
LED (#131)
WindowsBlinds (#22)

Bundle: HS322 Servo (#132)
Device-Type = Actuator
Bundle-Vendor = www.sensorplat
Bundle-Version = 1.1.0
Bundle-Activator = org.sensorplat
Bundle-SymbolicName = HS322Se
Ant-Version = Apache Ant 1.6.2

```
if(channelNum == 0) {
    if (vehiclePresent(sensorReading)) {
        // Open Door & turn on light
        servo.moveServo(OPEN);
        led.turnOnLED();
    }
}
if(channelNum == 1) {
    if (vehiclePresent(sensorReading)) {
        //Close Door & turn off light
        servo.moveServo(CLOSE);
        led.turnOffLED();
    }
}
```

Figure 8. Eclipse plug-in for programming smart spaces.

When implementing a new application, the IDE lets the programmers pick the sensors, actuators or existing applications that are relevant to the new project. It then automatically analyzes the dependency and configures the workspace such that the programmers can utilize these resources by invoking their public APIs.

The IDE also provides templates and hints to guide the placement of behavioral logics when creating a new application. This effectively eliminates the entry barrier for new programmers learning how to program OSGi bundles. It also reduces the burden on experienced programmers from repeatedly coding the same OSGi boilerplate source necessary to create a working bundle.

Once the implementation is complete, the IDE provides a one-button operation to compile the code, prepare the bundle jar file, and deploy the new application to the remote framework of the programmer's choice.

The tight integration with Eclipse gives programmers all the features expected from a modern IDE, such as method completion, automatic builds, and code refactorization support. The familiar interface and programming practice also make it easier for Java programmers to learn how to develop applications in pervasive computing environments.

6. Case Studies

The Atlas platform has already been used in a number of research projects, both within the University of Florida and in collaboration with other groups.

6.1. Gator Tech Smart House

The Gator Tech Smart House (GTSH) [1,2], as discussed in the first section and shown in Fig. 9, is a 2500 sq. ft. assistive environment for seniors. Originally constructed as an integrated pervasive space, the GTSH is now transforming into a showcase of programmable pervasive space technology. The Atlas platform is the foundation of this transformation.

Figure 9. The Gator Tech Smart House.

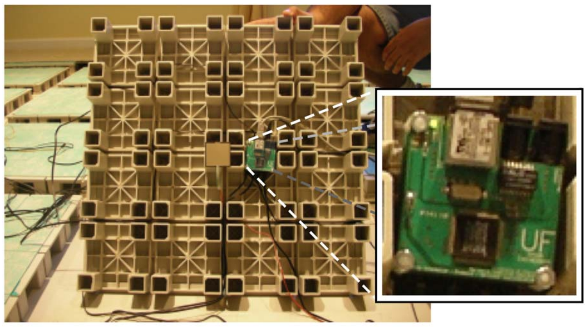


Figure 10. Smart Floor tile, with Atlas node.

Several applications in the GTSH have already been moved to the Atlas platform. These applications include the Smart Floor [22] (Fig. 10, a low-cost, unencumbered indoor location-tracking system), the servo-controlled window blinds, and the front door (keyless entry and security with door opener and electric deadbolt). Details about these projects can be found in [23]. The remaining applications, such as the Smart Plug [24], will soon be running on Atlas.

6.2. Purdue NILE-PDT

The NILE-PDT (Phenomena Detection and Tracking) system was developed by the Indiana Database Center at Purdue University to detect and track environmental phenomena (gas clouds, oil spills, etc.). They required a platform that would allow their system to sample data streams from many different sensors. Additionally, NILE-PDT needed to control the data streams by altering the sampling rate of the sensors using feedback algorithms, a mechanism that required uniform interfacing with every sensor in the network. The Atlas platform was a perfect match for NILE-PDT.

In addition to providing a uniform interface to heterogeneous sensors, Atlas also offers a plug-and-play development model, even for applications written outside our framework. NILE-PDT had been in development for years, and it was almost fully implemented before Atlas was available. Other conflicts arose during this collaboration, such as NILE-PDT using UDP for communication (the current Communication Layer for Atlas uses TCP) or the device drivers for the sensors providing raw data readings (NILE-PDT expects time-stamped data).

We expect these types of conflicts will be common when groups use a third-party platform with their existing applications. We created a proxy system in the framework to resolve these issues. Using this interface, the NILE-PDT developers were able to create a proxy in our framework that formed the bridge between the sensor services and the NILE-PDT engine. Our middleware allows external applications to upload and register these proxy services into the framework.

The NILE-PDT team was able to implement their system without having any knowledge about the internal workings of the Atlas platform. A collaborative paper [25] and demonstration based on this research was presented at the Very Large Data Bases 2005 conference.

7. Comparison

Table 1 compares Atlas and five other sensor platforms in terms of components, connectivity options, features offered, and commercial availability.

Table 1. Platform Comparison

	Atlas	*MICA2 Mote*	*Telos*	*Cube*	*MASS*	*Smart-Its*
Microcontroller	Atmega128L	Atmega128L	TI MSP430	Atmega128L	Cygnal C8051F125	PIC 18F6720
Flash memory	128K	128K	48K	128K	128K	128K
Communication protocols supported	Wired Ethernet, WiFi, Bluetooth, USB, Zigbee	RF	IEEE 802.15.4	RF	RF	RF
Service frameworks supported	OSGi, .NET (ongoing)	None	None	None	None	None
Ad-hoc network support	No	Yes	Yes	Yes	Yes	Yes
Modular	Yes	Partial	No	Partial	Yes	Partial
Commercially Available	Yes	Yes	Yes	No	No	Yes

8. Current Status & Upcoming Release

The Atlas platform is functional and is commercially available to researchers and other groups. It can be obtained from Pervasa, Inc. (a University of Florida startup) at www.pervasa.com. The current iteration of the platform includes a standardized Processing Layer, a choice of four Communication Layers and various Device Connection Layers.

The next version of Atlas is already in development. This section will provide an overview of the major changes and improvements that we have planned.

8.1. Hardware Roadmap

We are investigating two combined Power/Communication Layers using power-line communication and power-over-Ethernet. Other power-related development includes energy harvesting designs using resources such high-frequency RF, solar, or vibration.

The stackable 2x2" form factor has worked well for most projects, but we are also investigating a more compact design for applications that require a smaller footprint. Extreme miniaturization is not planned because working (programming, installing, etc.) with tiny nodes is difficult.

We are also developing pre-configured Device Connection Layers. These layers come with integrated sensor arrays, allowing users to deploy an Atlas network right out of the box.

8.2. Atlas Node Operating System

We are currently porting TinyOS [26], the de facto standard for sensor network operating systems, to our platform. This new operating system will facilitate the development of on-node applications, will provide a bootloader for over-the-network reprogramming if firmware updates are necessary, and will open the door to ad-hoc networking with Atlas.

8.3. Security

Security in sensor networks is being studied, but Atlas includes a middleware capable of acting as the central authority for the network, something not available in other sensor networks (which are usually ad-hoc). The Atlas security model will manage the authorization of devices in an environment, and ensure data entering the middleware framework is coming from the correct devices. We can also take advantage of the Context and Knowledge Representation modules of our middleware for programmable pervasive spaces (Fig. 1). The low-level encryption features will, as other researches have recommend [27], make use of elliptic curve cryptography due to the smaller keys necessary to provide equal protection as RSA. Key distribution in the network can be integrated into the plug-and-play configuration using our USB connection layer.

8.4. Privacy Preservation

Creating pervasive computing spaces such as assistive smart houses requires a rich sensor environment, and preserving the privacy of residents is critical for user acceptance. We are in the early stages of developing a privacy model that annotates the device driver bundles with information concerning acceptable use of the data that service provides (destroy after processing, local logging, transmit to authorized parties only, etc.). The middleware framework is responsible for ensuring applications that make use of a service cannot access any resource that could violate that service's privacy settings.

8.5. Data Processing

While the primary goal for the Atlas project was developing a sensor and actuator platform to enable programmable pervasive spaces, the platform is an appropriate tool for researchers and groups working in other fields. As mentioned in Sec. 6.2, we are collaborating with Purdue University on the Nile-PDT project, which involves reading streams of data from sensors. We are working to enhance the data streaming capabilities of the platform. This involves expanding the data processing functionality both onboard Atlas nodes and inside the service framework. Some of the capabilities being developed are data filtering, data aggregation, and query processing.

8.6. Distributed Middleware Servers

The current release of the Atlas platform assumes a single computer will be running the middleware framework that hosts the software-service representations of connected devices. However, in an extremely large or densely packed environment, so many services running on a single computer could result in poor performance. Additionally, a single pervasive space could cover many geographically dispersed areas. We are developing a distributed version of the middleware to solve these issues. This new architecture allows a hierarchical grouping of middleware servers, each of which can connect to Atlas nodes and other servers, feeding information to a parent server.

8.7. Alternate Service Frameworks

The Atlas platform is focused primarily on an OSGi-based service framework. However, we are in the process of porting our middleware to the Microsoft .NET framework. An alpha version of this framework exists, where devices are represented as Web Services.

9. References

[1] http://www.icta.ufl.edu/gt.htm

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

[3] S. Helal, "Programming pervasive spaces," in IEEE Pervasive Computing, vol. 4, no. 1, pp. 84-87, 2005.

[4] C. Lee, D. Nordstedt and A. Helal, "OSGi for Pervasive Computing," the Standards, Tools and Best Practice Department, IEEE Pervasive Computing, vol. 2, no. 3, Sept 2003.

[5] 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.

[6] H. Yang, E. Jansen, and H. Helal, "A comparison of two programming models for pervasive computing," Ubiq. Ntwk and Enablers to Context Aware Services, Intl. Symp. on Apps. and the Internet, January, 2006.

[7] http://robotics.eecs.berkeley.edu/~pister/SmartDust/

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

[9] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in Proceedings of the 4th Intl. Conf. on Information Processing in Sensor Networks, April, 2005.

[10] L. Nachman, R. Kling, J. Huang and V. Hummel, "The Intel mote platform: a Bluetooth-based sensor network for industrial monitoring," in 4th Intl. Conf. on Infor. Processing in Sensor Networks, April, 2005.

[11] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler and J. Anderson, "Wireless sensor networks for habitat monitoring," in 1st ACM Intl. Wksp on Wireless Sensor Networks and Apps., pp. 88-97, Sept. 2002.

[12] S. Greenberg and C. Fitchett, "Phidgets: easy development of physical interfaces through physical widgets," in 14th ACM Symp. on User Interface Software and Technology, pp. 209-218, Nov. 2001.

[13] B. O'Flynn et al., "The development of a novel minaturized modular platform for wireless sensor networks," in 4th Intl. Conf. on Information Processing in Sensor Networks (IPSN), April, 2005.

[14] N. Edmonds, D. Stark and J. Davis, "MASS: modular architecture for sensor systems," in 4th IPSN, April, 2005.

[15] R. Pon et al., "Networked infomechanical systems: a mobile embedded networked sensor platform," in 4th IPSN, April, 2005.

[16] D. Lymberopoulos and A. Savvides, "XYZ: a motion-enabled power aware sensor node platform for distributed sensor network applications," in 4th IPSN, April, 2005.

[17] C. Park, J. Liu and P. Chou, "Eco: an ultra-compacy low-power wireless sensor node for real-time motion monitoring," in 4th IPSN, April, 2005.

[18] H. Gellerson, G. Kortuem, A. Schmidt and M. Beigl, "Physical prototyping with Smart-Its," in IEEE Pervasive Computing, vol.3, no. 3, pp. 74-82, July-Sept 2004.

[19] A. Helal, W. Mann and C. Lee, "Assistive environments for individuals with special needs," Smart Environments, John Wiley and Sons, Inc., pp. 361-383, 2005.

[20] A. Helal et al., "Assistive environments for successful aging," in 1st Intl. Conf. on Smart homes and Health Telemetrics, pp. 104-112, Sept. 2003.

[21] K. Lee, "IEEE 1451: a standard in support of smart transducer networking," 17th Instrumentation and Measurement Technology Conf., vol. 2, pp. 525-528, May 2000.

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

[23] R. Bose, J. King, H. El-zabadani, S. Pickles and A. Helal, “Building Plug-and-Play Smart Homes Using the Atlas Platform,” 4th Intl. Conf. on Smart Homes and Health Telemetrics, June, 2006.

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

[25] M. Ali et al., "NILE-PDT: a phonemenon detection and tracking framework for data stream management systems," in Proceedings of the Very Large Data Bases Conf., Aug. 2005.

[26] http://www.tinyos.net/

[27] B. Arazi, I. Elhanany, O. Arazi and H. Qi, "Revisiting public-key cryptography for wireless sensor networks," in IEEE Computer, vol. 38, no. 11, pp. 103-105, Nov. 2005.