The Gator Tech Smart House: A Programmable Pervasive Space

Many first-generation pervasive computing systems lack the ability to evolve as new technologies emerge or as an application domain matures. Programmable pervasive spaces, such as the Gator Tech Smart House, offer a scalable, cost-effective way to develop and deploy extensible smart technologies.

Sumi Helal
William Mann
Hicham El-Zabadani
Jeffrey King
Youssef Kaddoura
Erwin Jansen
University of Florida

Research groups in both academia and industry have developed prototype systems to demonstrate the benefits of pervasive computing in various application domains. These projects have typically focused on basic system integration—interconnecting sensors, actuators, computers, and other devices in the environment.

Unfortunately, many first-generation pervasive computing systems lack the ability to evolve as new technologies emerge or as an application domain matures. Integrating numerous heterogeneous elements is mostly a manual, ad hoc process. Inserting a new element requires researching its characteristics and operation, determining how to configure and integrate it, and tedious and repeated testing to avoid causing conflicts or indeterminate behavior in the overall system. The environments are also closed, limiting development or extension to the original implementers.

To address this limitation, the University of Florida's Mobile and Pervasive Computing Laboratory is developing *programmable pervasive spaces* in which a smart space exists as both a runtime environment and a software library.[1] Service discovery and gateway protocols automatically integrate system components using generic middleware that maintains a service definition for each sensor and actuator in the space. Programmers assemble services into composite applications, which third parties can easily implement or extend.

The use of service-oriented programmable spaces is broadening the traditional programmer model. Our approach enables domain experts—for example, health professionals such as psychiatrists or gastroenterologists—to develop and deploy powerful new applications to users.

In collaboration with the university's College of Public Health and Health Professions, and with federal funding from the National Institute on Disability and Rehabilitation Research (NIDRR), we are creating a programmable space specifically designed for the elderly and disabled. The Gator Tech Smart House in Gainesville, Florida, is the culmination of more than five years of research in pervasive and mobile computing. The project's goal is to create *assistive environments* such as homes that can sense themselves and their residents and enact mappings between the physical world and remote monitoring and intervention services.

SMART HOUSE TECHNOLOGIES

Figure 1 shows most of the "hot spots" that are currently active or under development in the Gator Tech Smart House. An interactive 3D model available at www.icta.ufl.edu/gt.htm provides a virtual

Published by the IEEE Computer Society

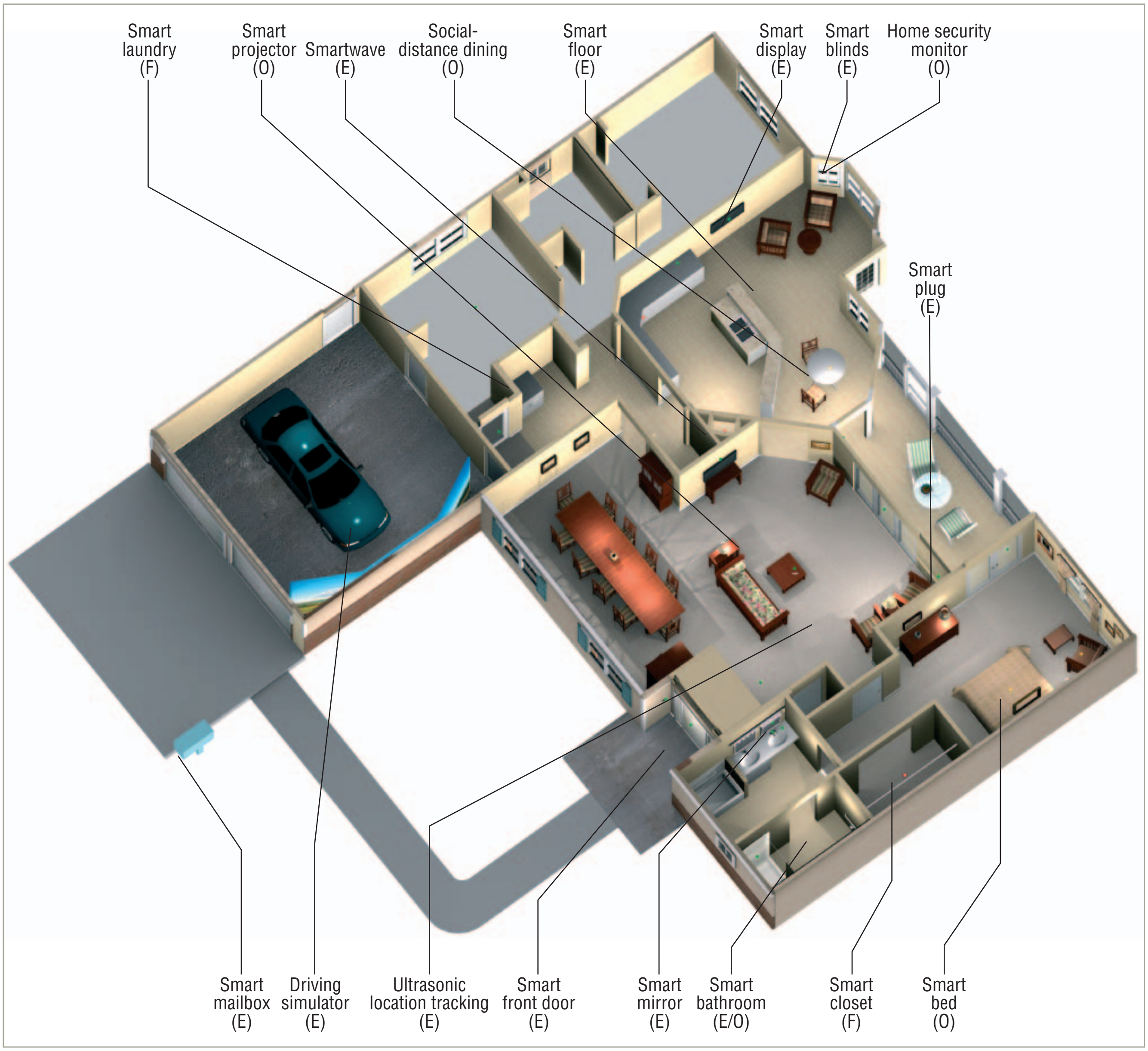


Figure 1. Gator Tech Smart House. The project features numerous existing (E), ongoing (O), or future (F) "hot spots" located throughout the premises.

tour of the house with up-to-date descriptions of the technologies arranged by name and location.

Smart mailbox. The mailbox senses mail arrival and notifies the occupant.

Smart front door. The front door includes a radio-frequency identification (RFID) tag for keyless entry by residents and authorized personnel. It also features a microphone, camera, text LCD, automatic door opener, electric latch, and speakers that occupants can use to communicate with and admit visitors.

Driving simulator. The garage has a driving simulator to evaluate elderly driving abilities and gather data for research purposes.

Smart blinds. All windows have automated blinds that can be preset or adjusted via a remote device to control ambient light and provide privacy.

Smart bed. The bed in the master bedroom has special equipment to monitor occupants' sleep patterns and keep track of sleepless nights.

Smart closet. The master bedroom closet will, in the future, make clothing suggestions based on outdoor weather conditions.

Smart laundry. In combination with the smart closet, future RFID-based technology will notify residents when to do laundry as well as help sort it.

Smart mirror. The master bathroom mirror displays important messages or reminders—for example, to take a prescribed medication—when needed. This technology could be expanded to other rooms.

Smart bathroom. The master bathroom includes a toilet paper sensor, a flush detector, a shower that regulates water temperature and prevents scalding, and a soap dispenser that monitors occupant cleanliness and notifies the service center when a refill is required. Other technologies under development measure occupant biometrics such as body weight and temperature.

Smart displays. With the display devices located throughout the house, e entertainment media and information can follow occupants from room to room.

Smartwave. The kitchen's microwave oven automatically adjusts the time and power settings for any frozen food package and shows users how to properly prepare the food for cooking.

Smart refrigerator/pantry. A future refrigerator will monitor food availability and consumption, detect expired food items, create shopping lists, and provide advice on meal preparation based on items stored in the refrigerator and pantry.

Social-distant dining. Occupants will be able to use Immersive video and audio technologies installed in the breakfast nook to share a meal with a distant relative or friend.

Smart cameras. Image sensors monitor the front porch and patio for privacy and security.

Ultrasonic location tracking. Sensors, currently installed only in the living room, detect occupants' movement, location, and orientation.

Smart floor. Sensors in the floor, currently only in the kitchen and entertainment center area, identify and track the location of all house occupants. We are also developing technologies to detect when an occupant falls and to report it to emergency services.

Smart phone. This "magic wand for the home" integrates traditional telephone functions with remote control of all appliances and media players in the living room. It also can convey reminders and important information to home owners while they are away.

Smart plugs. Sensors behind selected power outlets in the living room, kitchen, and master bedroom detect the presence of an electrical appliance or lamp and link it to a remote monitoring and intervention application.

Smart thermostats. In the future, occupants will be able to personalize air conditioning and heat settings throughout the house according to daily tasks or context—for example, they could slightly increase the temperature when taking a shower on a cold winter night.

Smart leak detector. Sensors in the garage and kitchen can detect a water leak from the washing machine, dishwasher, or water heater.

Smart stove. This future device will monitor stove usage and alert the occupant, via the smart bed, if the stove has been left on.

Smart projector. We are developing a projector that uses orientation information provided by ultrasonic location tracking and displays cues, reminders, and event notifications to the living room wall that the occupant is currently facing.

Home security monitor. A security system under development continually monitors all windows and doors and, upon request, informs the resident whether any are open or unlocked.

Emergency call for help. A future system will track potential emergencies, query the resident if it suspects a problem, and issue a call for outside help when necessary.

Cognitive assistant. Another system under development guides residents through various tasks and uses auditory and visual cues to provide reminders about medications, appointments, and so on.

MIDDLEWARE ARCHITECTURE

To create the Gator Tech Smart House, we developed a generic reference architecture applicable to any pervasive computing space. As Figure 2 shows, the middleware contains separate physical, sensor platform, service, knowledge, context management, and application layers. We have implemented most of the reference architecture, though much work remains to be done at the knowledge layer.

Physical layer

This layer consists of the various devices and appliances the occupants use. Many of these are found in a typical single-family home such as lamps, a TV, a set-top box, a clock radio, and a doorbell. Others are novel technologies such as the smartwave and the keyless entry system adapted to the Smart Home's target population.

Sensors and actuators such as smoke detectors, air conditioning and heating thermostats, and security-system motion detectors are part of the physical layer as well. In addition, this layer can include any object that fulfills an important role in a space, such as a chair or end table.

Sensor platform layer

Not all objects in a given space can or should be accounted for. For example, it may be desirable to capture a toaster, which could cause a fire if inadvertently left on, but not a blender. Each sensor platform defines the boundary of a pervasive space within the Smart House, "capturing" those objects attached to it. A sensor platform can communicate with a wide variety of devices, appliances, sensors, and actuators and represent them to the rest of the middleware in a uniform way.

A sensor platform effectively converts any sensor or actuator in the physical layer to a software

A sensor platform effectively converts any sensor or actuator in the physical layer to a software service that can be programmed or composed into other services.

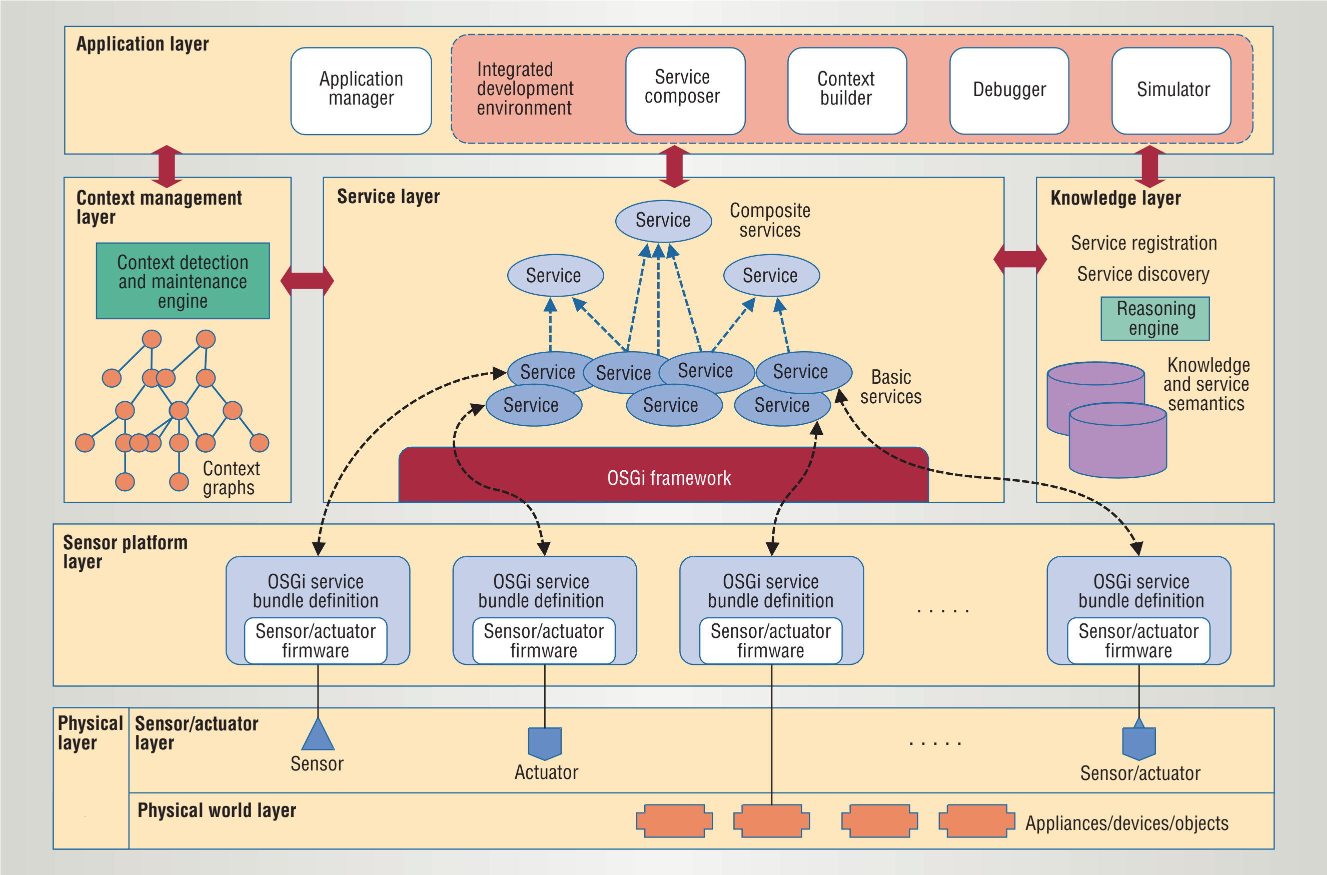


Figure 2. Smart space middleware. ***This generic reference architecture is applicable to any pervasive computing environment.***

service that can be programmed or composed into other services. Developers can thus define services without having to understand the physical world. Decoupling sensors and actuators from sensor platforms ensures openness and makes it possible to introduce new technology as it becomes available.

Service layer

This layer contains the Open Services Gateway Initiative (OSGi) framework, which maintains leases of activated services.

Basic services represent the physical world through sensor platforms, which store service bundle definitions for any sensor or actuator represented in the OSGi framework. Once powered on, a sensor platform registers itself with the service layer by sending its OSGi service bundle definition.

Application developers create *composite* services by using a service discovery protocol to browse existing services and using other bundle services to compose new OSGi bundles. Composite services are essentially the applications available in the pervasive space.

A set of de facto *standard* services may also be available in this layer to increase application developers' productivity. Such services could include voice recognition, text-to-speech conversion, scheduling, and media streaming, among many others.

Knowledge layer

This layer contains an ontology of the various services offered and the appliances and devices connected to the system. This makes it possible to reason about services—for example, that the system must convert output from a Celsius temperature sensor to Fahrenheit before feeding it to another service.

Service advertisement and discovery protocols use both service definitions and semantics to register or discover a service. The *reasoning engine* determines whether certain composite services are available.

Context management layer

This layer lets application developers create and register contexts of interest. Each context is a graph implemented as an OSGi service wire API linking various sensors together. A context can define or restrict

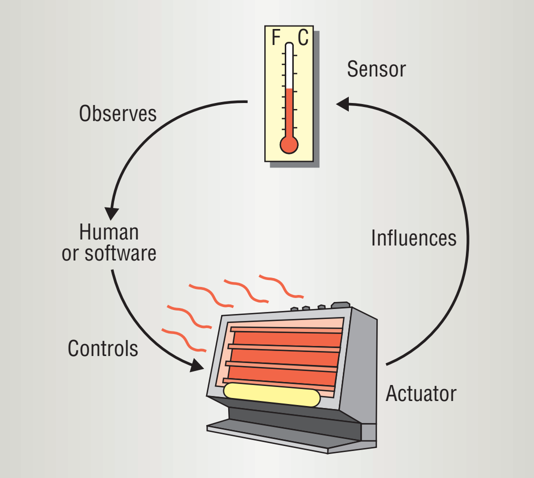


Figure 3. Sensor and actuator interaction. Actuators influence sensors, which observe the state of the world and can in turn cause the system or a user to activate the actuator.

service activation for various applications; it can also specify states that a pervasive space cannot enter.

The *context engine* is responsible for detecting, and possibly recovering from, such states. Our reference architecture has no fixed context-aware programming model.

Application layer

This layer consists of an application manager to activate and deactivate services and a graphical-based integrated development environment with various tools to help create smart spaces. With the *context builder* a developer can visually construct a graph that associates behavior with context; a programmer also can use it to define impermissible contexts and recovery services. In addition, developers can use the *service composer* to browse and discover services as well as compose and register new ones. Other tools include a debugger and simulator.

CONTEXT AWARENESS

Programming an intelligent space such as the Gator Tech Smart House involves three distinct activities:

- Context engineering—interpreting sensory data and identifying high-level states of interest such as "hot" and "sunny."
- Software engineering—describing the various software components' behavior—for example, turning on the heater or generating a possible menu from a set of ingredients.
- Associating behavior with context—defining which pieces of software can execute in a particular context and which pieces the system should invoke upon a contextual change.

Critical to this process is the observe-control interaction between sensors and actuators, as shown in Figure 3.

Abstracting sensory data

The Smart House obtains information about the world through various sensors and can use this data to undertake certain actions. The typical home likewise relies on sensors to effect changes—for example, if it gets too cold, the thermostat will activate the heater. However, what distinguishes a truly robust context-aware system such as the Smart House is the ability to abstract state information and carry out actions that correspond to these high-level descriptions.[2,3]

Most sensors are designed to detect a particular value in one domain. For example, a temperature sensor might determine that it is 95 degrees Fahrenheit in the house, or a light sensor might record 10,000 lux of light coming through the window. However, hard-coding behavior for each possible combination of direct sensor values is difficult to implement, debug, and extend.

It is far easier to associate actions with abstractions such as "hot" and "sunny," which encompass a range of temperature and luminescence values. When it is hot, the system turns on the air conditioning; if it is sunny outside and the television is on, the system closes the blinds to reduce glare. This approach can easily be extended to various contexts—for example, if the resident is on a diet, the system could prevent the smartwave from cooking a greasy pizza.

Context management

In addition to sensors, the Smart House consists of actuators—physical devices with which people can interact. An actuator can change the state of the world. Sensors, can, in turn, observe an actuator's effect. For example, a light sensor might determine that the house or resident turned on a lamp. Based upon the observed state of the world, the house or resident might activate an actuator.

Every actuator in the Smart House has a certain *intentional effect* on a domain, which a sensor that senses that particular domain can observe. For example, the intentional effect of turning on the heater is to increase the temperature.

Given a clear description of an actuator's intentional effect, it is possible to determine acceptable behaviors for a given context by examining all possible behaviors in the current state and identifying which intentional effects are mutually exclusive. This guarantees, for example, that the system will never invoke the air conditioning and heater simultaneously.

Context changes can occur due to

- an actuator's intentional effect—for example, after turning on the heater, the house temperature goes from "cold" to "warm;" or
- a natural or otherwise uncontrollable force or event—for example, the setting sun causes a change from "daytime" to "nighttime."

Ideally, a smart space that enters an impermissible context should try to get out of it without human monitoring. Toward this end, we are exploring ways that will enable the Smart House to learn how to invoke a set of actuators based upon state information to automatically self-correct problems.

Given a standardized description of an actuator's intentional behavior in a certain domain and how a sensor value relates to a particular context, it should be possible to determine which actuator to invoke to escape from an impermissible context. If escape is impossible, the system can inform an external party that assistance is required. For example, if the pantry does not contain any food and no grocery-delivery service is available, the system could inform an outside caregiver that it is time to restock.

SENSOR PLATFORM

Integration can become wieldy and complex due to the various types of sensors, software, and hardware interfaces involved. Consider, for example, climate control in a house. Normally, you would have to hard-wire the sensors to each room, connect these sensors to a computer, and program which port on the computer correlates to which sensor. Further, you must specify which port contains which type of sensor—for example, humidity or temperature.

To systematically integrate the various devices, appliances, sensors, and actuators and to enable the observe-control loop in Figure 3, we created a sensor platform that represents any attached object in a pervasive space simply as a Java program—more specifically, as an OSGi service bundle.

To control climate in a home, for example, you would install a wireless sensor platform node in each room, connect both a humidity sensor and temperature sensor to each node, and program the firmware for each node. In addition to the firmware, the sensor platform nodes would contain the sensor driver that decodes temperature and humidity data.

Simply powering up a sensor node causes it to transmit the driver wirelessly to a surrogate node, such as a home PC, where the sensors are immediately accessible via other applications. The PC would require no configuration or hardware interfacing. The sensor driver is surrogate software—Java byte code that contains static information about the sensor and the services it provides—stored in an electrically erasable programmable read-only memory (EEPROM) on the sensor platform node. The platform itself does not understand or process the code; rather, it processes the firmware and other low-level C programs that send data between the sensor and platform.

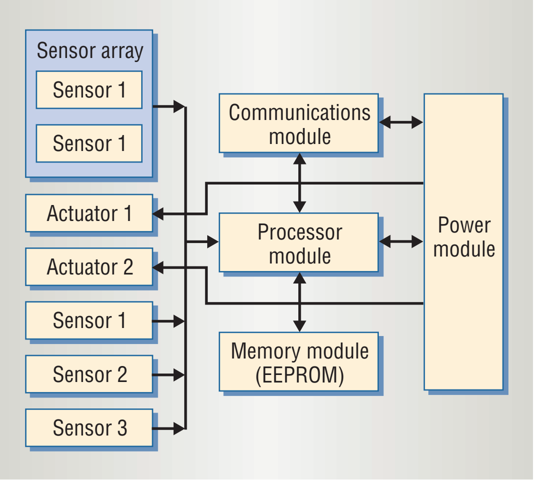


Figure 4. Sensor platform architecture. The modular design provides for alternative and flexible configurations.

The individual node architecture shown in Figure 4 is modular and provides for alternative and flexible configurations. We use a stackable design to connect alternative memory, processor, power, and communication modules.

The *memory module* provides a mechanism for easily modifying an EEPROM store used for read and write capabilities on the node. This storage contains bootstrap data that specifies general sensor and actuator information.

The *processing module* currently uses an 8-bit Atmel ATmega 128 processor. The processor is housed on a board that is optimized for low power consumption and has two RS232 ports, a Joint Test Action Group (IEEE 1149) and ISP port, and more than 50 programmable I/O pins. We are developing alternative modules with more powerful processing capability, including an onboard Java virtual machine.

The *communication module* currently uses RF wireless communication with a simple transmission protocol. We are also testing and debugging a 10BaseT Ethernet module utilizing a simplified IPv4 stack. Future modules will support low-power Wi-Fi and power-line communication. The latter will also connect to an alternative *power module.*

When a sensor platform is powered up, its EEPROM data acts as a bootstrap mechanism that provides the larger system—for example, a network server or home PC—with the information and behavioral components required to interact with a specific device, appliance, sensor, or actuator. The data can be specified as either human-readable (XML, text with a URL, and so on) or machine-readable (for example, Java byte code) depending on the specific application. In addition to byte code, stored data includes device-specific information such as the manufacturer's name, product serial number, and sensor type.

SMART PLUGS

Creating a scalable self-sensing space is impractical using existing pervasive computing technologies.[4] Most smart appliances available in the market today do not contain a controllable interface. In addition, numerous available protocols are incompatible. For example, the X10 protocol offers an easy, affordable way to turn a house into a smart one, but many smart devices are not X10 enabled. Regardless of the technology used, a smart space should be able to communicate with any new smart device.[5,6]

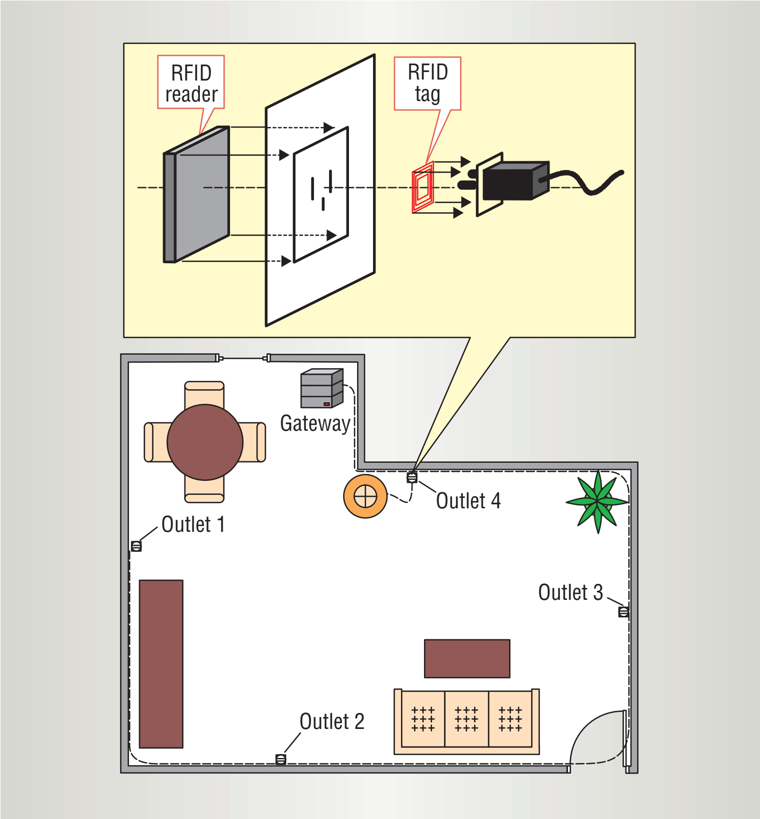


Figure 5. Smart plugs. Each power outlet is equipped with a low-cost RFID reader connected to the main computer, while each electrical device has an RFID tag attached to the plug's end with information about the device.

To address this problem, we have developed smart plugs, which provide an intelligent way to sense electrical devices installed in an intelligent space. As Figure 5 shows, each power outlet in the Gator Tech Smart House is equipped with a low-cost RFID reader connected to the main computer. Electrical devices with power cords, such as lamps and clocks, each have an RFID tag attached to the plug's end with information about the device. When a user plugs the device into an outlet, the reader reads the tag and forwards the data to the main computer.

OSGi bundles represent new devices to be installed in the smart space. A bundle is simply a Java archive file containing interfaces, implementations for those interfaces, and a special Activator class.[7] The jar file contains a manifest file that includes special OSGi-specific headers that control the bundle's use within the framework.

Each RFID tag has user-data-allocated memory that varies from 8 to 10,000 bytes. Depending on the size of its memory, the tag itself could contain the entire OSGi bundle representing the new device. If the bundle is too large, the tag could instead contain a referral URL for downloading the gateway software from a remote repository. The referral URL can use any protocol that the gateway server has access to, such as http and ftp. Using a Web server also makes upgrading the bundle as easy as replacing the software.

The gateway bundles installed in the framework perform all the required downloading and installation of the gateway software for the individual bundles. When a user installs a new device, the system downloads each bundle and registers it in the OSGi framework. Upon request, the framework can report a list of installed devices, all of which can be controlled via methods available in the bundle. In this way, the framework enacts a mapping between the smart space and the outside world.

Figure 6 shows a user—for example, a service technician at a monitoring center—controlling a lamp in the Smart House via a remote application; a click on the lamp will download all available methods associated with this device. When the user clicks on a method, the remote application sends a request to the gateway to execute the action.

SMART FLOOR

In designing the Gator Tech Smart House floor, we wanted to deploy a low-cost, accurate, unencumbered, position-only location system that could later serve as the foundation for a more powerful hybrid system. Drawing on extensive location-tracking and positioning research, we initially experi-

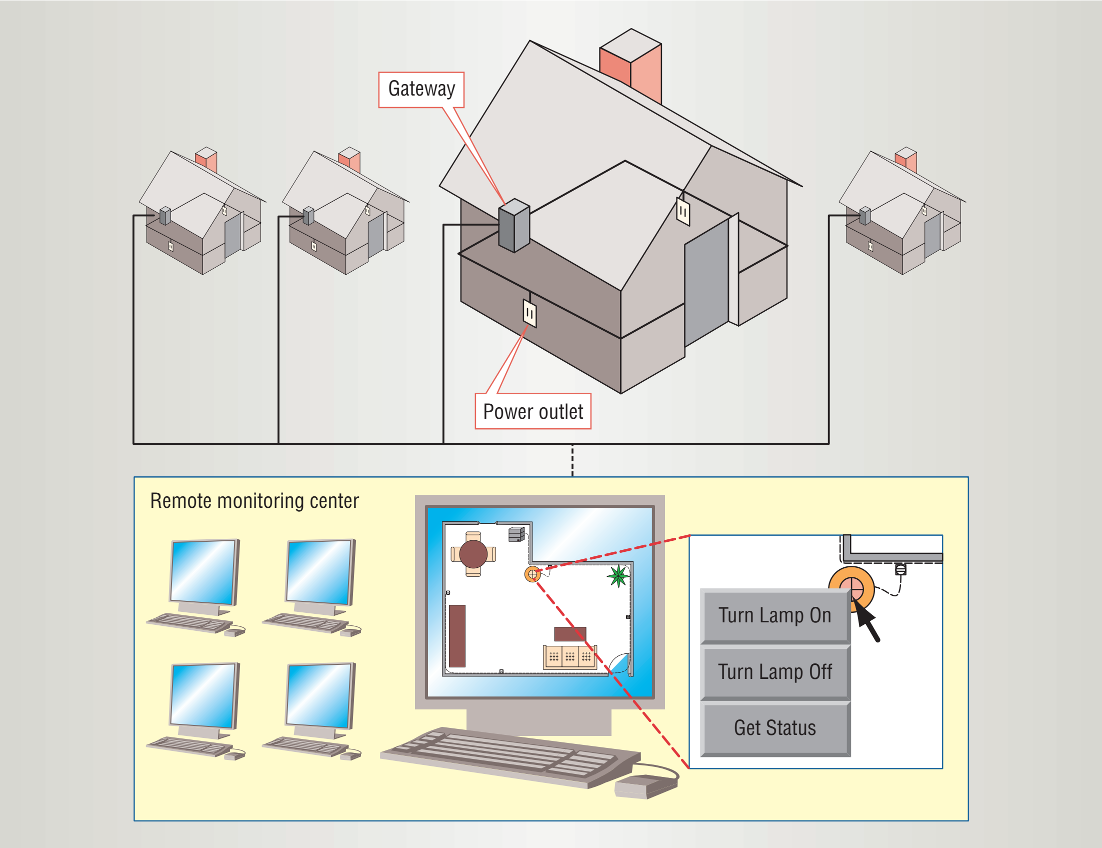


Figure 6. Remote monitoring of electrical appliances. Clicking on a method causes the remote application to send a request to the Smart House gateway to execute the action.

mented with an acoustic-based location system. Using a set of ultrasonic transceiver pilots in the ceiling, the master device would regularly send chirps into the environment. Users wore vests in which transceivers tags attached to the shoulders would listen for the chirp and respond with their own.

While this technology provides precise user position and orientation measurements, it was inappropriate for the Smart House. Each room would require a full set of expensive pilots, and residents would have to don special equipment, which is extremely intrusive and defeats the desired transparency of a pervasive computing environment.[8,9]

Instead, we opted to embed sensors in the floor to determine user location.[10-12] The benefit of not encumbering users outweighed the loss of orientation information, and the availability of an inexpensive sensor platform made this solution extremely cost-effective.

We had been using Phidgets (www.phidgetsusa.com) for various automation tasks around the Smart House. The Phidgets Interface Kit 8/8/8 connects up to eight components and provides an API to control the devices over a Universal Serial Bus. Each platform also integrates a two-port USB hub, making it easy to deploy a large network of devices. We created a grid of 1.5-inch pressure sensors under the floor, as shown in Figure 7, and connected this to the existing Phidgets network.

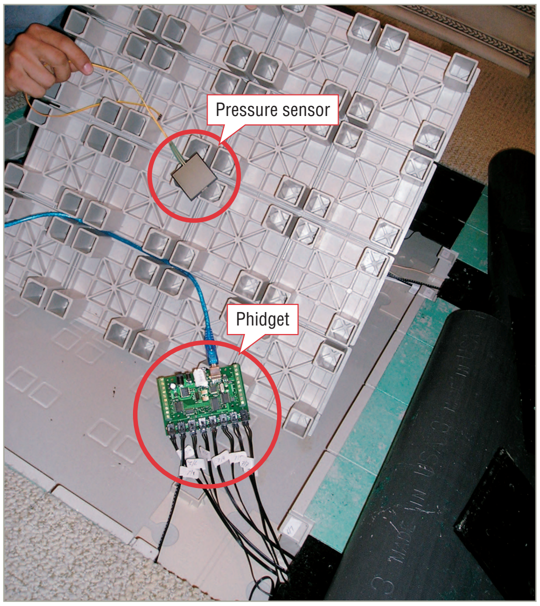


Figure 7. Smart floor tile block. The Smart House floor consists of a grid of 1.5-inch pressure sensors connected to a network of Phidgets.

Table 1. Smart floor deployment costs in the kitchen, nook, and family room.

Number of blocks	Sensors/ block	Sensor platform/block	Sensor unit price	Sensor platform unit price	Total cost	Cost/ square foot
64	1	1/8	$10	$95	$1,400	$4

The smart house has a 2-inch residential-grade raised floor comprised of a set of blocks, each approximately one square foot. This raised surface simplified the process of running cables, wires, and devices throughout the house. In addition, the floor's slight springiness puts less strain on the knees and lower back, an ergonomic advantage of particular interest to seniors.

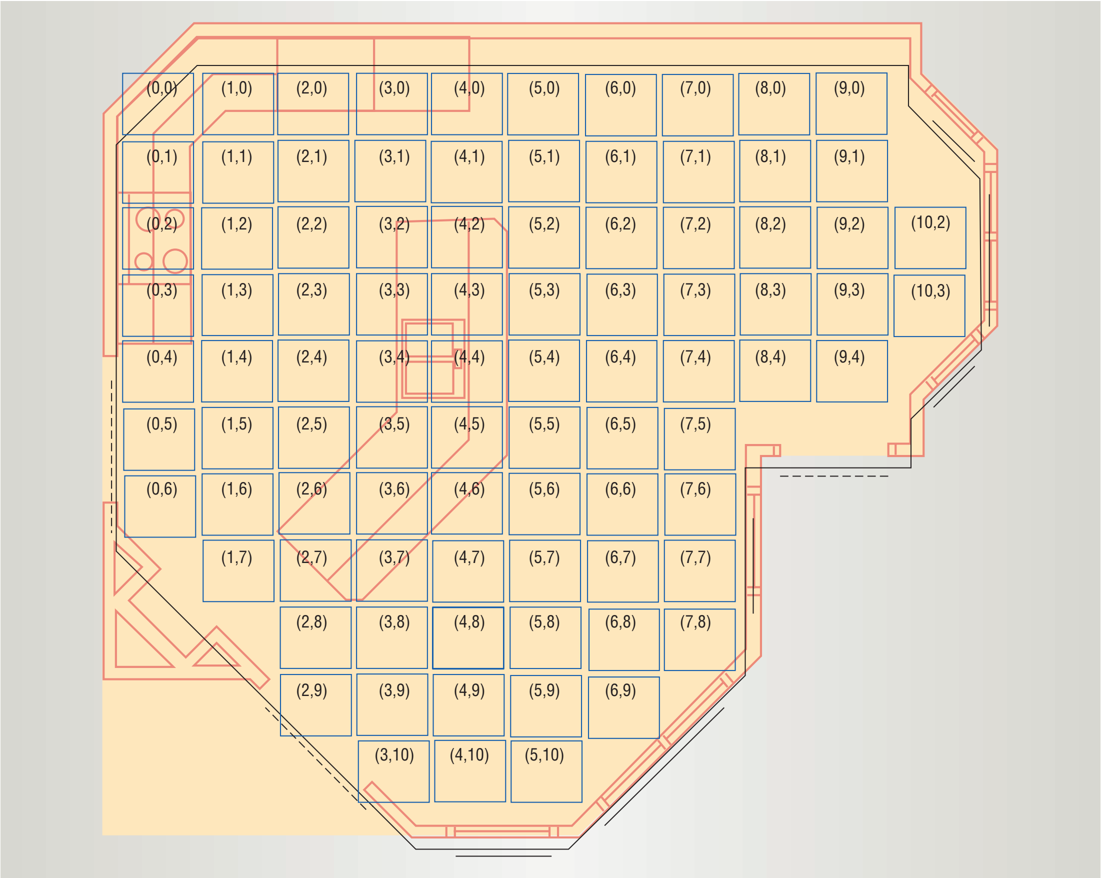
We discovered another, unexpected benefit of the raised surface: It allows us to greatly extend the pressure sensors' range. When a person steps on a tile block, the force of that step is distributed throughout the block. A single sensor at the bottom center can detect a footstep anywhere on that block. In fact, we had to add resistors to the sensor cables to reduce sensitivity and eliminate fluctuations in the readings.

Table 1 details the costs of deploying the smart floor in the kitchen, nook, and family room, a total area of approximately 350 square feet. We do not have to factor the price of the raised floor, which is comparable to other types of residential flooring, into our cost analysis because it is a fundamental part of the Smart House and is used for various purposes.

The hardest part of deploying the smart floor involved mapping the sensors to a physical location. Installing the sensors, labeling the coordinates, and manually entering this data into our software took approximately 72 person-hours.

Figure 8 shows the mapping system we used for the kitchen, nook, and family room. Tiles with solid lines represent blocks with sensors underneath, while those with dotted lines indicate gaps in cov-

Figure 8. Smart floor mapping system. Tiles with solid lines represent blocks with sensors underneath, while those with dotted lines indicate gaps in coverage due to appliances or room features.



erage due to appliances or room features such as cabinets or the center island.

In the future, we intend to redeploy the smart floor using our own sensor platform technology, which will include spatial awareness. This will greatly simplify the installation process and aid in determining the location of one tile relative to another. We will only need to manually specify the position of one tile, and then the system can automatically generate the mapping between sensors and physical locations.

Pervasive computing is rapidly evolving from a proven concept to a practical reality. After creating the Matilda Smart House, a 900-square-foot laboratory prototype designed to prove the feasibility and usefulness of programmable pervasive spaces as assistive environments, we realized that hacking hardware and software together resulted in some impressive demonstrations but not something people could actually live in.

We designed the second-generation Gator Tech Smart House to outlive existing technologies and be open for new applications that researchers might develop in the future. With nearly 80 million baby boomers in the US just reaching their sixties, the demand for senior-oriented devices and services will explode in the coming years. Ultimately, our goal is to create a "smart house in a box": off-the-shelf assistive technology for the home that the average user can buy, install, and monitor without the aid of engineers. ■

Acknowledgments

We thank the NIDRR for the generous funding that made this research possible. Many students and research assistants contributed to the Gator Tech Smart House project, including James Russo, Steven Van Der Ploeg, Andi Sukojo, Daniel Nieten, Steven Pickels, Brent Jutras, and Ed Kouch. Dr. Choonhwa Lee of Hanyang University significantly helped us to conceptualize the reference architecture.

References

1. S. Helal, "Programming Pervasive Spaces," *IEEE Pervasive Computing,* vol. 4, no. 1, 2005, pp. 84-87.
2. A.K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing,* vol. 5, no. 1, 2001, pp. 4-7.
3. G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research," tech. report TR2000-381, Dept. of Computer Science, Dartmouth College, 2001.
4. R.K. Harle and A. Hopper, "Dynamic World Models from Ray-tracing," *Proc. 2nd IEEE Int'l Conf. Pervasive Computing and Communications,* IEEE CS Press, 2004, pp. 55-66.
5. H-W. Gellerson, A. Schmidt, and M. Beigl, "Adding Some Smartness to Devices and Everyday Things," *Proc. 3rd IEEE Workshop Mobile Computing Systems and Applications,* IEEE CS Press, 2000, pp. 3-10.
6. H. Gellersen et al., "Physical Prototyping with Smart-Its," *IEEE Pervasive Computing*, vol. 3, no. 3, 2004, pp 74-82.
7. D. Marples and P. Kriens, "The Open Services Gateway Initiative: An Introductory Overview," *IEEE Comm. Magazine,* vol. 39, no. 12, 2001, pp. 110-114.
8. J. Hightower and G. Borriello, "Location Systems for Ubiquitous Computing," *Computer,* Aug. 2001, pp. 57-66.
9. G. Welch and E. Foxlin, "Motion Tracking: No Silver Bullet, But a Respectable Arsenal," *IEEE Computer Graphics and Applications,* vol. 22, no. 6, 2002, pp. 24-38.
10. M.D. Addlesee et al., "The ORL Active Floor," *IEEE Personal Comm.*, vol. 4, no. 5, 1997, pp. 35-41.
11. H.Z. Tan, L.A. Slivovsky, and A. Pentland, "A Sensing Chair Using Pressure Distribution Sensors," *IEEE/ASME Trans. Mechatronics,* vol. 6, no. 3, 2001, pp. 261-268.
12. R.J. Orr and G.D. Abowd, "The Smart Floor: A Mechanism for Natural User Identification and Tracking," *Proc. Human Factors in Computing Systems* (CHI 00), ACM Press, 2000, pp. 275-276.

Sumi Helal *is a professor in the Department of Computer and Information Science and Engineering at the University of Florida and is director and principle investigator of the Mobile and Pervasive Computing Laboratory. His research interests include pervasive and mobile computing, collaborative computing, and Internet applications. Helal received a PhD in computer science from Purdue University. He is a senior member of the IEEE and a member of the ACM and the Usenix Association. Contact him at helal@cise.ufl.edu.*

William Mann *is a professor and chairman of the Department of Occupational Therapy at the University of Florida and is director of the Rehabilitation Engineering Research Center. His research focuses on aging and disability, with an emphasis*

on compensatory strategies to maintain and promote independence. Mann received a PhD in higher education from the University of Buffalo. He is a member of the American Society on Aging, the American Geriatric Society, and the Gerontological Society of America. Contact him at wmann@phhp.ufl.edu.

Hicham El-Zabadani *is a PhD student in the Department of Computer and Information Science and Engineering at the University of Florida and is a member of the Mobile and Pervasive Computing Laboratory. His research interests include self-sensing spaces, computer vision, and remote monitoring and intervention. El-Zabadani received an MS in computer science from the Lebanese American University. Contact him at hme@cise.ufl.edu.*

Jeffrey King *is a PhD student in the Department of Computer and Information Science and Engineering at the University of Florida and is a member of the Mobile and Pervasive Computing Laboratory. His research interests include security in pervasive computing systems, context-aware computing, thermodynamically reversible computing, and real-time graphics rendering. King received an MS in computer engineering from the University of Florida. He is a member of the ACM. Contact him at jeffking@cise.ufl.edu.*

Youssef Kaddoura *is a PhD student in the Department of Computer and Information Science and Engineering at the University of Florida and is a member of the Mobile and Pervasive Computing Laboratory. His research interests include indoor location tracking and location- and orientation-aware pervasive services. Kaddoura received an MS in computer science from the Lebanese American University. Contact him at yok@cise.ufl.edu.*

Erwin Jansen *is a PhD candidate in the Department of Computer and Information Science and Engineering at the University of Florida and is a member of the Mobile and Pervasive Computing Laboratory. His research interests include programming models for pervasive computing, context awareness, artificial intelligence, and peer-to-peer systems. Jansen received an MS in computer science from Utrecht University. Contact him at ejansen@cise.ufl.edu.*