A Demonstration of Pervasive Device Integration with SEAP-based Middleware

Drew Stovall, Seth Holloway, Jorge Lara-Garduno, Christine Julien
The University of Texas at Austin
Mobile and Pervasive Computing Group
Austin, TX, USA
{dstovall, sethh, jlara-garduno, c.julien}@mail.utexas.edu

ABSTRACT

In this paper, we describe a demonstration of the SEAP middleware architecture applied to pervasive computing applications. SEAP, or **S**ensor **E**nablement for the **A**verage **P**rogrammer, is an architectural pattern specifically targeted at junior and hobbyist level programmers. It builds on existing knowledge and technology resources commonly available to this target audience, and provides a friendly environment to create customized applications that interact with the physical world. While we discuss some of the motivation behind our work and give a brief overview of the SEAP architecture, the majority of this paper describes a proposed demonstration of the technology. This interactive demonstration is designed to be accessible to a wide variety of people, and to spur discussions on middleware for end-users of pervasive computing.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Software Architectures ; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Standardization

Keywords

SEAP, sensors, demonstration

1. INTRODUCTION

Pervasive computing technology has found its way into the military, industry, and even our homes. Devices are required to work together to perform useful tasks that are simply not feasible without cooperation. However, the cooperation that we see between devices is almost invariably predefined by manufacturers. Implementation details are often inaccessible or require an immense amount of product-specific knowledge. These factors limit the ability of end-users to customize the behaviors of devices and their interactions in pervasive environments.

The SEAP middleware architecture [8] creates a simple coordination framework to allow novice programmers to write programs that collect data and issue commands to remote devices, allowing them to quickly develop valuable personalized applications that rely on device coordination.

Currently, sensors are hard to use; they use proprietary communication protocols, unique data formats, and a variety of programming languages. The combination of these effects keep sensors from their deserved place in the wild. Imagine an aware home scenario where a user wants to lower his electric bill by enabling home automation to proactively adjust light levels as inhabitants move around the building. While the concept is simple in theory, there are few realizations in practice. Lights may be motion controlled using specialized hardware, but this represents a closed solution that a user cannot modify. Similar systems use RFID, but they are also closed. To integrate more features, custom installations are possible, but they are prohibitively expensive and likely use specialized hardware. Instead of using an expensive, proprietary system, we propose an easily understood software approach that uses open standards and well-known languages. In addition, our approach will allow the home's inhabitants to easily tailor applications' behavior to their particular requirements and preferences.

While SEAP is broadly applicable, characteristics of the approach favor specific applications. Because SEAP is optimized for approachability rather than performance, we envision SEAP being applied to rapid prototyping and heterogeneous device integration for projects such as automation, monitoring, and latency-tolerant command and control; example domains include smart spaces, aware homes, and intelligent construction sites. SEAP applications are quickly created, easily customized, and extensible. Disparate data sources are easily aggregated and processed which opens the wide world of sensors to developers that would otherwise be excluded by the exotic programming languages and other intimidating road-blocks.

In the next section we discuss related work. In Section 3 we briefly describe the SEAP approach. Section 4 details how we propose to demonstrate SEAP, and how conference participants will interact with the display. As requested, in Section 5 we specify some equipment that we require. A brief conclusion is offered in Section 6.

Middleware'08 Companion December 1-5, 2008 Leuven, Belgium

2. RELATED WORK

The notion of sensor and web integration is not entirely new, and several related projects have paved the way for our proposed approach. CoolTown allows networked mobile devices to publish data on the web through a variety of tailored protocols [2]. Data being published includes information about a device's characteristics (location, for example), enabling a degree of content-based discovery. Another ambitious project has created a centralized website that will accept sensor data generated worldwide [4], a technique cleverly titled *slog* (sensor log). Here, sensors first communicate their data to a base station supporting a particular sensor network. The base station funnels sensor data to the clearinghouse using a connection to the Internet. Because we are interested in enabling more personal applications, in SEAP, users control their own data. This provides a more distributed computing approach since data and actuation events are only shared relative to a local space.

Other current approaches to sharing sensor data build on standard web services using SOAP, WSDL, and XML. The Open Geospatial Consortium (OGC) allows governments and affiliated companies to take advantage of a massive set of federated devices and sensors using SensorML, a sensor model language that defines an XML schema to identify and use sensors [3]. Microsoft's SenseWeb project [11] has also provided a generic method to push sensor data online. However, both approaches rely on SOAP web services, which can be inflexible, slow, hard to maintain and manage, and heavyweight [9]. For large deployments, economies of scale marginalize the overhead, but SOAP web services present an unnecessary cost for small deployments that we set out to alleviate. This is especially relevant to ubiquitous computing deployments where devices are often resource-constrained, demanding efficient and streamlined solutions.

While rooted in different technologies, there are a number of other designs to reduce the efforts required to develop pervasive computing applications. For example, Weis et al. [13] use visual programming techniques to reduce the learning curve typically required. Other approaches such as [7] and [1] provide additional layers of abstraction to manage complexities that can be hidden from the developer. We anticipate that these techniques to be complementary to the SEAP middleware architecture, and might be combined to further ease software development for pervasive computing.

In the SEAP approach we minimize the interface for both devices and programmers by relying on a simple but expressive form for data movement, HTTP GET and POST commands. Our approach is consistent with representational state transfer (REST) principles [6] in an effort to be lightweight, flexible, and compatible. REST is an architectural style for network systems that promotes the transmission of domain-specific data over HTTP. Key to REST is the notion of resources; every piece of specific data should be universally accessible as a resource. Users interact with resources using a small set of well-defined commands to manipulate the representation of a resource. Some work has been done to apply the REST style in the sensor domain [5], however, this work violates many REST principles with large, complex systems that require a great deal of configuration and knowledge; this reduces benefits innate in the initial REST proposal. SEAP, on the other hand, approaches the problem with a minimalist perspective: get the data online in a form that entry level web programmers can already use.

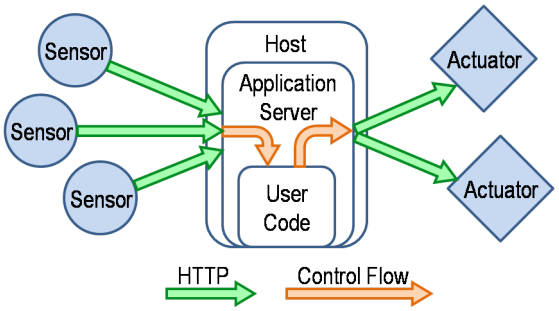

Figure 1: SEAP data flow among devices and user.

By using standard HTTP, we inherit the benefits of both past and future work on HTTP and allow programmers to begin writing ubiquitous applications immediately.

3. SEAP ARCHITECTURE

The SEAP architecture addresses the challenge of custom coordination by exploiting the widespread knowledge of basic website programming. At a high level SEAP extends standard web-programming techniques by replacing users with devices. This simple substitution allows us to build on the vast teaching- and knowledge-base already available to the web-programming community, as well as various technologies already built and maintained by various organizations.

In the deployment of a typical web application, a programmer installs their code in an application server. The application server handles the details of HTTP, thread management, and a number of other complex issues that are orthogonal to the application itself. Thus, when clients interact with an application, their requests are presented to the programmer in tidy, high-level data structures. To support a variety of languages, there are a number of COTS application servers each allowing the programmer to apply any existing programming skills.

In the deployment of a SEAP application, the programmer also uses an application server to deploy their code. Instead of web application clients, a SEAP application handles requests from pervasive computing devices. Some of these requests provide data to the application in much the same way a form is submitted in a web application. Other devices request new configuration or commands, an analog to the web application's personalized content. A digram of how the data flows in a SEAP application is shown in Figure 1. In this Figure, data is passed between the devices and the application server using HTTP. Since the user application is hosted by the application server, data is passed by simply passing control to the appropriate function.

The architecture described here serves as the basic system, but a number of extensions (which mirror web programming advances) will allow even simpler application development. For example, people create blogs using services hosted on remote machines. Pervasive computing coordination applications hosted remotely can likewise provide server support and domain specific programming "languages".

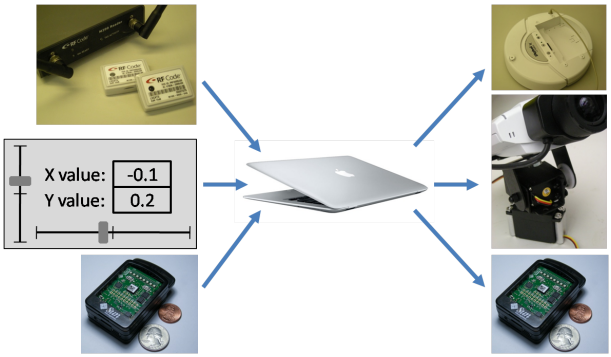

Figure 2: Flow of data among devices in demonstration. On the left are *sensors*, on the right are *actuators*, and in the center is the SEAP *server*.

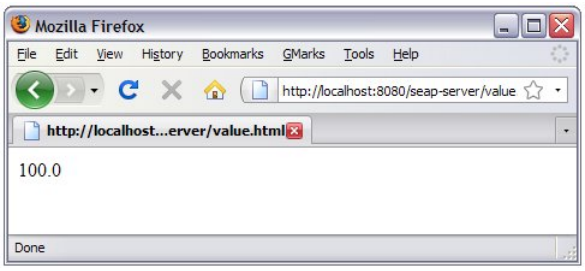

Figure 3: A web-browser will be able to read any advanced SEAP server output.

4. THE DEMONSTRATION

To demonstrate the novel features of our middleware architecture and how it is used to create new applications, we will use the devices typical of a pervasive computing environment (e.g. a user's home). A laptop computer will host the example applications during the demonstration; applications will receive data from a subset of devices and send data to a different subset of devices. An overview of the demonstration equipment is shown in Figure 2. In this section, we will detail the specific devices that will be present and their capabilities. We will then talk about how conference participants will interact with the demonstration on site.

4.1 Sensors

For the purposes of this description, we will refer to any device providing data to the SEAP server as a *sensor*. For example, a Sun SPOT device [12] provides readings taken from its internal 3-dimensional accelerometer. The device will provide three floating point numbers representing the relative direction of gravity to the SPOT's own coordinate system. Using these values, a SEAP application can discern how the device is tilted. We will talk about how this value is used later in this section.

To represent more abstract contextual values, we will also provide an active RFID system which will periodically send presence information to the SEAP server. Using a small Faraday cage, we will simulate the movement of tagged items or personnel in a monitored area. Several tags will be available so that applications may build on the interplay between multiple tags.

To provide the ability to model very diverse and rich source data for the SEAP system, we will also be providing a *virtual* sensor in the form of a small GUI application. This sensor will provide a number of user controls to provide numerical and boolean data to the SEAP server, as well as free-form area where participants may provide their own (character-based) data. We anticipate using this functionality to replicate any example applications provided by spectators.

4.2 Actuators

To show how pervasive computing might affect users' environments, we will provide a number of devices that are capable of acting on data provided by a SEAP application. We will will refer to these devices as *actuators*. As one example, a Roomba Create robot will request commands from the SEAP server. Using only two numerical outputs, speed and turning angle, a SEAP application will be able to steer the robot remotely.[1]

The Sun SPOT devices mentioned in the previous subsection will be providing sensor data to the SEAP server. However, the devices also provide eight tri-color LEDs. The device will periodically request color and intensity information and update the LEDs appropriately. Additionally, since the SEAP server is making values available to HTTP clients, a simple web-browser will also be able to display the server's values as shown in Figure 3. Like the virtual sensor above, this will allow us to demonstrate the SEAP approach to any application domain.

Another possible actuator included in the demonstration is a camera with pan-and-tilt capabilities. The camera's output will be displayed for participants to view while the servos will be controlled by the SEAP server.

4.3 The SEAP Server

At the center of the demonstration, we will provide a laptop computer hosting application servers in at least two programming languages (Java & PHP). While only one application server can run at a given time, we can switch between them quickly. Each server will be pre-populated with applications to demonstrate the SEAP approach. Attendees will be able to interact with the demonstration both passively and actively. They will be able to manipulate the sensors and view their effects on the actuators. The source code for the example applications will also be available for inspection and manipulation. Specifically, we will allow attendees to update the applications on-site to realize new behaviors and interactions.

The software running on the sensors and actuators themselves will not be modifiable (as in a real SEAP deployment) which protects the integrity of the demonstration. No matter what changes are made to the SEAP server, the sensors will continue feeding data to it and the actuators will continue reading commands. If a bug is introduced into the SEAP server, the example application can be restored and the demonstration returned to normal.

[1]The current implementation uses a long data cable to control the robot. We anticipate the completion of an on-board controller which will allow untethered operation.

4.4 Example Applications

Here is a brief list of applications that we will provide as examples or create on-site with attendees:

- *Spot-to-Bot:* As the Sun SPOT is tilted forwards and backwards, the Roomba robot will drive faster and slower. Like-wise, tilting the SPOT left and right will effect the robot's steering. One application might interpret the SPOT's input as speed while another might interpret it as acceleration, a simple conversion to make on the demonstration floor.
- *Presence monitor:* As Active RFID Tags are moved in and out of range of the reader, the SEAP server will instruct a Sun SPOT to alter the color of its LEDs. This will model a simple In-Out board like the one described in [10]. There are a number of variations on this theme that attendees can explore on-site.
- *Energy efficient home:* Using the SunSPOT LEDs to model the activity of a home's air-conditioning system, we can set up an application that only turns the air conditioner on when an occupant is home and the measured temperature is above a given threshold. We will use the RFID system to monitor occupancy and the current temperature will be provided by the virtual sensor.

5. REQUIRED EQUIPMENT

As described, the demonstration requires the following items in addition to those we will supply. However, if some of these items are not available, we are convinced that we can still provide an interesting demonstration a subset of the functionality described above.

- *Power, four outlets:* Much of the equipment is battery powered, but we will require at least four outlets.
- *Table:* To lay out the devices and provide a work surface for participants, a table about two meters long should be sufficient.
- *Floor space:* The Roomba robot will be moving about on the floor. We do not anticipate requiring any special considerations for this, but want the organizers aware of it.

6. CONCLUSION

This paper details a demonstration of the SEAP middleware architecture for pervasive computing environments. With the SEAP middleware, hobbyists will be able to develop meaningful, custom applications without learning the low-level details of sensors and sensor networks. We hope that showing both premeditated and improvised examples will clearly show how simple these applications are to build given the proper support.

7. ACKNOWLEDGMENTS

The authors would like to thank the Center for Excellence in Distributed Global Environments and the Pervasive Computing Test Bed for providing research facilities and the collaborative environment. This research was funded in part by NSF Grant #CNS-0626777 and AFOSR Grant #FA9550-07-1-0157. The views and conclusions herein are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

8. REFERENCES

[1] F. J. Ballesteros, E. Soriano, G. Guardiola, and K. Leal. Plan B: Using files instead of middleware abstractions. *IEEE Pervasive Computing*, 6(3):58–65, July-September 2007.

[2] J. Barton, T. Kindberg, H. Dai, and N. Priyantha. Sensor-enhanced mobile web clients: an XForms approach. *WWW*, pages 80–89, 2003.

[3] M. Botts. Sensorml. `http://vast.uah.edu`, 2007.

[4] K. Chang, N. Yau, M. Hansen, and D. Estrin. SensorBase.org-A Centralized Repository to Slog Sensor Network Data. *DCOSS/EAWMS*, 2006.

[5] W. Drytkiewicz, I. Radusch, S. Arbanowski, and R. Popescu-Zeletin. pREST: a REST-based protocol for pervasive systems. *MASS*, pages 340–348, 2004.

[6] R. Fielding and R. Taylor. Principled design of the modern Web architecture. *TOIT*, 2(2):115–150, 2002.

[7] R. Handorean, J. Payton, C. Julien, and G.-C. Roman. Coordination middleware supporting rapid deployment of ad hoc mobile systems. In *Proceedings of the 1st International Workshop on Mobile Computing Middleware, co-located with ICDCS 2003*, pages 362–368, May 2003.

[8] S. Holloway, D. Stovall, A. Dalton, and C. Julien. So many sensors, so little data. In *Proceedings of the First International Workshop on Software Architectures and Mobility (SAM '08)*, Leipzig, Germany, 10 May 2008.

[9] C. Kohlhoff and R. Steele. Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems. *WWW*, pages 03–2002, 2003.

[10] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '99)*, pages 434–441, New York, NY, USA, 1999. ACM.

[11] A. Santanche, S. Nath, J. Liu, B. Priyantha, and F. Zhao. SenseWeb: Browsing the Physical World in Real Time. *Demo Abstract, IPSN*, 2006.

[12] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White. Java on the bare metal of wireless sensor devices. In *Proceedings of the 2nd International Conference on Virtual Execution Environments*, June 2006.

[13] T. Weis, M. Knoll, A. Ulbrich, G. Muhl, and A. Brandle. Rapid prototyping for pervasive applications. *IEEE Pervasive Computing*, 6(2):76–84, April–June 2007.