Simplifying the Programming of Intelligent Environments

by

Seth Michael Holloway, B.S., M.S.

Dissertation Proposal
Software Engineering Track
The Department of Electrical and Computer Engineering
The University of Texas at Austin

Committee Members
Prof. Randolph Bias
Prof. Christine Julien (Supervisor)
Prof. Sarfraz Khurshid
Prof. Miryung Kim
Prof. Dewayne E. Perry

The University of Texas at Austin
September 2009
Abstract

In the future, computers will be everywhere: carried by everyone and integrated into the environment. The increased computation and communication capabilities will enable environments to react to occupants through automated decision-making. On the path to intelligent environments, there are a number of hard problems. First and foremost, devices (sensors and actuators) must be made more approachable for average users. This dissertation aims to make applications for these intelligent environments easier to develop, thereby increasing the adoption and prevalence of emerging sensing and actuation technologies. Existing approaches to application development for these pervasive computing networks require detailed knowledge of low-level programming languages and embedded hardware. These current approaches also require adoption of new tools and languages that are not user-friendly. Intelligent environments will not be commonplace until average people can not only set up the necessary hardware and software but can also tailor the embedded software to personalized applications. This dissertation endeavors to create a software framework for pervasive computing networks that is easy to adopt and use.

In resolving these challenges, this dissertation focuses first on programmers then extends to end-users. We start by standardizing device communication using highly available and widely accepted protocols. To ease the development effort for programmers, we allow complex sensor and actuator applications to be viewed as
web applications. We term this work SEAP, or Sensor Enablement for the Average Programmer. SEAP makes available sensor data and communication capabilities that will allow average programmers to develop intelligent environment applications. However, to develop the massive amount of software necessary for personalized intelligent environments, it is also necessary to allow end-users to tailor the behavior of their sensors and actuators. This dissertation’s second major contribution, Sensor Enablement for End-Users (SEEU) will provide a visual programming interface that will allow users to create personalized automated behaviors given available devices and data. This will require a principled design guided by end-users. Combining an intuitive interface and a middleware for sensor interactions, SEEU will be an end-user programming framework that will allow average people to create useful applications for their intelligent environments. For pervasive computing to truly bloom in these envisioned intelligent environments, it must be simple to develop new software for the plethora of potential available hardware. Upon completion, this dissertation will offer a software solution necessary to match the proliferation of hardware. The total package will usher in a new world of possibilities in pervasive computing.
Contents

Abstract ii

Chapter 1 Introduction 1
  1.1 Existing Approaches 3
  1.2 Solution 4
  1.3 Intended Contributions 5
  1.4 Impact 5

Chapter 2 SEAP Into It 7
  Sensor Enablement for the Average Programmer
  2.1 Sensor Enablement In Context 10
  2.2 SEAP Onto the Web 12
    2.2.1 Underlying Infrastructure 13
    2.2.2 SEAP Architecture: Devices 14
    2.2.3 SEAP Architecture: Application Server 19
  2.3 SEAP-Based Implementations 21
    2.3.1 SEAP Applied to an Existing Application 21
    2.3.2 SEAP From Scratch 24
  2.4 SEAP Evaluation 25
    2.4.1 Study Tasks 26
Chapter 3  I SEEU—the World Should Too

Sensor Enablement for End-Users

3.1 The Latest in a Long Line

3.2 User Driven Design: The SEEU Approach

3.2.1 Learning How Users Think... About Intelligent Environments

3.2.2 Developing an Effective Interface

3.2.3 Putting it all together

3.3 Can End-Users SEEU?

3.4 Remaining Tasks

3.5 What Happens When We All SEEU

Chapter 4  Research Timeline

Chapter 5  Conclusion

Appendices

Appendix A  Appendix

A.1 SEEU Smart Home Survey

A.1.1 Recruitment Letter

A.1.2 Cover Letter

A.1.3 Survey Text

Bibliography
Chapter 1

Introduction

Sensors and actuators are currently too complex and misunderstood to be widely adopted [34, 40, 65]. Proprietary programming languages, communication protocols, and data formats compound to create a complex domain that few people can understand. This reality is at odds with the power that sensors and actuators can provide for all people. Grand visions of intelligent environments, spaces that automatically react to current conditions, predate personal computers.

Intelligent environments powered by sensors and actuators have been part of the public consciousness for several decades, but the dream is not yet a reality. In 1951 popular science fiction author Ray Bradbury described the Happylife Home, a residence that took care of its inhabitants [19]. In 1991, Mark Weiser espoused the idea of being immersed in technology and brought ubiquitous computing to researchers [77]. In the many years since Weiser described his vision for sensors and actuators embedded in the environment, researchers have successfully tackled a number of open issues. Along the way hardware and software technologies have evolved tremendously, and we are now within reach of user programmable intelligent environments. Sensors—everything from thermometers to iPhones—are now common in homes and offices. Actuators are also increasingly common; for example,
many appliances are shipping with remote control possibilities. The next step is to open the devices to use by their owners. Research solutions exist now [43, 44], but these solutions involve proprietary protocols and custom implementations. Few programmers and even fewer laypeople can truly harness the capabilities of these devices due to proprietary languages and a lack of standards.

Modern descriptions of intelligent environments have evolved along with technological capabilities and societal needs: sensors and actuators are due to revolutionize many parts of our lives including environmental monitoring, construction management, and healthcare. With talk about global warming, people are much more interested in data about their environment like carbon dioxide output, general air quality, and various temperatures. By embedding devices in their homes, people can monitor their consumption and then make informed decisions about how to live a more ecologically friendly life. Similarly, businesses could reduce electricity consumption by controlling heating, cooling, and lights automatically. An intelligent construction site would allow the foreman to track valuable assets, including materials and crew members, thereby increasing safety and reducing theft. The foreman could also easily keep track of properties that he would otherwise have to check in person; properties such as the structural integrity of a building, the time for concrete to dry, etc. Home healthcare will be vital with a massive aging population. Aided by a handful of sensors in the home, elderly people could retain a level of care similar to a nursing home while still enjoying the freedom and dignity of living on their own. Aging-in-place is only one piece of the home healthcare puzzle. By arming people with cheap medical sensors and allowing them to program their residences, everyone could benefit from routine (constant) check-ups. Diagnoses will be made with more data, and emergency situations can be averted, or at worst, discovered quickly.
1.1 Existing Approaches

Given the incredible possibilities, it is no surprise that people are currently working to make these visions a reality. There are a variety of ways to reduce the complexities of sensors and actuators. Current approaches fall into two broad categories: using existing hardware or redesigning the infrastructure. The former category focuses on mainstream devices with sensing capabilities, such as using the iPhone [20], or formalizing device capabilities, such as by creating an all-encompassing ontology [17, 62]. The latter category proposes sweeping changes to existing communication protocols, hardware, and software to support the researchers’ proposed solutions [27, 52].

The push to use existing sensors and actuators, like the iPhone, is an excellent idea and is proving quite successful; however, an entire home cannot be automated using only the inhabitants’ cellular phones because cellular phones do not have the sensors and actuators necessary for proposed intelligent environments. Additionally, a home is unlikely to contain hundreds of cellular phones so cellular phones cannot accommodate many pervasive computing applications that require a multitude of devices distributed across the environment.

Others have chosen to formalize sensors with metalanguages and ontologies that make the devices easier to understand and use [17, 62]. This approach has been successful in large, targeted deployments, but it still leaves a great deal of overhead to the device users. For example, the United States Government can agree on standards and implement them internally without concern for conditions that consumers will face such as price, availability, and general usability.

Those redesigning infrastructure face several hurdles with usability and adoption. While it may be helpful to redesign the global communication (like the Internet), it is unlikely that sweeping changes would ever be adopted—especially not by average users. Sensors and actuators introduce several new concepts so it is impor-
tant for new methods to align with existing approaches to reduce the amount users have to learn before being comfortable. Creating new protocols would force users to understand new communication schemes as well as embedded devices.

1.2 Solution

This dissertation aims to reduce the complexities currently found in sensors and actuators making them easier to integrate into applications. Because sensors and actuators are poorly understood by the majority of the population, we will first translate the problem from the unknown sensor domain to a well understood domain. Given the current prevalence and future visions of the Internet, modeling intelligent environments as web applications is a logical choice. There is a wealth of information already available for web applications, and many programmers already know a web language such as PHP, Java, or Python. Once the problem is in a well understood domain, developing intelligent environments will be tractable for a much larger population of programmers.

To further reduce the complexities, we will focus on people in general, that is, end-users. To empower end-users we will build a usable graphical interface atop the solid programming foundation. There are over 55 million end-user programmers (people with no formal education or experience in the computer sciences) currently developing and using programs, for example using Microsoft Excel [64]. End-users are able to program whenever the interface is sufficiently simple and the end goal is sufficiently worthwhile. Initial research shows that people are excited about automating their environments and many users share a mental model of the process.
1.3 **Intended Contributions**

This proposal addresses the challenges related to widespread deployment of intelligent environments. Specifically, I propose to make the following contributions:

- **RT1.** Define a unified sensor/actuator model that makes existing and future devices usable by *average programmers*.

- **RT2.** Create a working embodiment of the sensor/actuator model as a middleware, SEAP (Sensor Enablement for the Average Programmer).

- **RT3.** Evaluate the usefulness of SEAP on a selection of novice software developers.

- **RT4.** Discover how end-users think about a tangible intelligent environment, smart homes, and how they express desired computer-controlled home behaviors.

- **RT5.** Design a programming interface that is accessible to end-users for creating customized intelligent environment behaviors.

- **RT6.** Create a software system, SEEU, that allows end-users to specify behaviors without in-depth knowledge about the underlying technologies.

- **RT7.** Evaluate the effectiveness of SEEU on end-users.

1.4 **Impact**

Upon the successful completion of the work proposed herein, pervasive computing applications will be easier to understand and develop. Programmers will have the full range of expression provided by sensors and actuators—while using their favorite language. End-users too will be able to program their own intelligent environment...
behaviors. The foundations provided by this dissertation will seamlessly accommodate device evolutions, allowing users to focus on behaviors and upgrade their system as technology and personal funds allow.

From an industrial standpoint, device manufacturers will have a simple specification to create usable, interoperable devices. Intellectual property and competitive advantages are maintained while users gain a simpler interface. This dissertation represents a necessary step towards standardization for the embedded device industry that currently offers immense capabilities through myriad complex, individual methods.

In addition, this work can help focus future research. The work blends software engineering, hardware specification, and usability. Others can adopt the approach and explore alternatives, particularly by creating domain-specific user interfaces.

We will first propose a system, Sensor Enablement for the Average Programmer (SEAP), that will easily allow “average programmers” to create pervasive computing applications from scratch and quickly integrate devices into existing applications. SEAP is covered in detail in Chapter 2. In Chapter 3 we will propose an end-user programming framework, Sensor Enablement for End-Users (SEEU), that will open the power of pervasive computing to everyone. SEEU will be designed with “average people” in each step of the development process. Chapter 4 provides a timeline for the completion of this work, and Chapter 5 concludes.
George is an undergraduate in Computer Science. After completing some college courses, he has learned many programming concepts and basic Java. George is an average programmer.

Recently, George has started reading about intelligent environments—an intriguing new idea that would make his life easier. George wants to use his Java programming skills to automate his own apartment. He hopes to start small and continue to augment the system as money allows. George’s initial goal is to automatically turn on lights in his current location. Later, he can play music, control the A/C and fans, open and close blinds, brew coffee, and all kinds of other neat additions. George is really excited! With a little bit of programming and a few pieces of hardware, he will never have to use a pesky light switch again!

George begins by searching for solutions, but, after extensive research, George gives up because nothing satisfies his requirements. There are inexpensive single-
purpose solutions like a motion sensitive light switch [51], but these are black-box replacements that cannot be altered or extended. To achieve his initial goal of illuminating only the occupied room, George could replace his light switches with the motion sensitive version; however, the motion sensitive light switch will not allow George to achieve his future goals of, for example, turning on the inhabitant’s favorite radio station. At the other end of the spectrum, George finds costly, complex, multi-purpose solutions like those from Crestron [2]. Using the Crestron suite of products, George could automate his entire apartment and control the site with a state-of-the-art remote; unfortunately, it would cost thousands of dollars and require extensive knowledge about Crestron products. Even worse, George would be locked into Crestron products that communicate using an undocumented protocol. George realizes the aware home reality is much more complicated than buying a few new components and writing some Java code.

George understands the application logic perfectly (“if someone is present then turn on lights”), but he does not know how to use the necessary pervasive computing elements: sensors and actuators, like motion sensors and lighting controls. George wants a cheap, extensible solution that he can program using a language he already knows.

While George may be fictional, this scenario is real. There are many “average programmers” with domain knowledge who would develop intelligent environments if they could easily interact with the necessary sensors and actuators, and they will all encounter the same problems when developing an expressive intelligent environment. Only highly-trained developers and sensor researchers have the knowledge necessary to create intelligent environments. The potential for pervasive computing research to impact the world is real, but not until the hardware is usable by average programmers like George.

Existing approaches are stovepipe solutions that solve a single problem, in-
dependent of other devices already deployed on the site. Current approaches fuse the entire application stack, creating unnecessary interdependencies. For example, in the case of the motion sensitive light switch, George cannot easily reuse the motion sensor from his light switch; in the case of the Crestron complete home system, George cannot integrate generic sensors into his intelligent apartment. Interacting with a simple device like the aforementioned motion sensor (that only has two states: motion or no motion) would require specialized knowledge of the unique communication protocol. Combining multiple devices would require knowledge of the many specialized protocols and languages as well as proficiency in an intermediary language that could accept the varied inputs and generate distinct outputs. Many programmers understand the basic application logic; they simply lack the depth currently necessary to develop intelligent environments. To enable programmability of pervasive computing applications, we propose a paradigm shift that empowers the average programmer rather than highly specialized device programmers.

We are interested in answering the following research question: “How can we empower average programmers to develop their own intelligent environments?” To answer this question we will complete the following three research tasks:

- **RT1.** Define a unified sensor/actuator model that makes existing and future devices usable by average programmers.

- **RT2.** Create a working embodiment of the sensor/actuator model as a middleware, SEAP (Sensor Enablement for the Average Programmer).

- **RT3.** Evaluate the usefulness of SEAP on a selection of novice software developers.

We classify this work as sensor enablement for the average programmer, or SEAP [38, 39, 69]. We cover previous efforts by other researchers related to SEAP
in Section 2.1 before going into our approach. The SEAP architecture combines sensors, actuators, an application server, and web programming languages; details are in Section 2.2. We have used SEAP to develop new applications and to evolve existing web applications into ubiquitous computing applications. We detail these implementations in Section 2.3. To further evaluate SEAP, we will perform a user study with “average programmers.” We discuss the user study in Section 2.4. In Section 2.5 we cover the remaining tasks. Finally, we discuss the ramifications of success in Section 2.6.

2.1 Sensor Enablement In Context

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 [15]. Data published includes information about a device’s characteristics (e.g., location), enabling a degree of content- and context-based discovery. Another project created a centralized website that accepts sensor data generated worldwide [22], a technique cleverly titled slog (sensor log). Sensors communicate their data to a base station that funnels sensor data to a clearinghouse. Because we are interested in enabling more personal applications, it is imperative that 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 access to sensors using SensorML, a sensor-specific language that defines an XML schema to use sensors [18]. Microsoft’s SenseWeb project [63] also provided a generic method to push sensor data online. Both approaches rely on SOAP web services, which can be inflexible, slow, hard to maintain and manage, and heavy-
weight [47]. SOAP web services present an unnecessary cost for small deployments that we set out to alleviate. This is especially relevant to pervasive computing deployments where resource-constrained devices often demand efficient, 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. [76] use visual programming techniques to reduce the learning curve typically required for creating new pervasive computing applications. Other approaches [14, 36] provide additional layers of abstraction to manage complexities that can be hidden from the developer. We anticipate that these techniques will be complementary to the SEAP middleware architecture and might be combined to further ease software development for pervasive computing.

Our goal is to reduce the apparent complexity of devices and enable programmers to develop pervasive computing applications. We do this by relying on a simple, expressive method to transfer data between disparate devices, standard HTTP. Our approach is consistent with representational state transfer (REST) principles [31] in an effort to be lightweight and flexible. REST is an architectural style that promotes the transmission of domain-specific data over standard HTTP. Users interact with resources using a small set of well-defined commands to manipulate the resource. Some work has been done to apply REST to pervasive computing [28, 52], however, this work violates many REST principles by requiring a great deal of configuration and a priori knowledge; this reduces benefits innate in the initial REST proposal. Perhaps more worrisome, these approaches propose altering the ubiquitous HTTP protocol, which would require patching every Internet-enabled device. Instead, we believe the solution must follow a minimalist approach: get the data online in a form that entry level web programmers can already use. The solution should also follow REST principles to inherit the benefits of both past and future work on HTTP.
Recent advances in cloud computing [21] have generated research similar to ours; this field is called the “web of things.” A number of projects from academia and open-source are attempting to make all devices usable online, such as JXTA-C [72], the RESTful plug and play experience [68], putting things to REST [78], the RESTlet [24], Sun Cloud APIs [54], and RESTful web services specification [60]. These projects share our goal (getting devices online with a simple interface) and use similar technologies (HTTP), but none solve the same problem in the same way that we do. Our research provides a well-defined interface focused on average programmers.

2.2 SEAP Onto the Web

In this section, we describe the SEAP approach in detail. Specifically, we present the necessarily simple software architecture that underlies the sensing, actuation, communication, and interaction capabilities as we approach our first research task: RT1. *Define a unified sensor/actuator model that makes existing and future devices usable by average programmers.*

By relying on well-established programming standards, SEAP brings the seemingly unapproachable task of programming pervasive computing applications into the hands of domain programmers who are experts in their applications’ requirements. SEAP hides complexities associated with data collection and actuator command with familiar web programming patterns, using lightweight software components deployed on resource-constrained devices to manage the distributed coordination tasks. Through SEAP’s abstractions, a developer can quickly tailor device and network configurations to a particular task by adjusting parameters in the application. At the same time, participating remote devices can use standard posting procedures to exchange sensor data and actuation commands in simple formats. A broad overview of the SEAP architecture can be seen in Figure 2.1. Here, “Server”
refers to a centralized host that may or may not be connected to the Internet.

Figure 2.1: A high-level view of the SEAP architecture.

We divide our description of the SEAP architecture into two aspects: the behavior of devices participating in a SEAP supported application, and the application server that hosts the applications. Before discussing these topics, we will briefly discuss the underlying infrastructure.

2.2.1 Underlying Infrastructure

The SEAP architecture relies on HTTP to communicate. To help the reader understand SEAP interactions, we will describe the basics as seen on the Internet. While SEAP applications do not have to be online, the ability to seamlessly integrate with the Internet is a huge benefit. The Internet is a network of computers connected using TCP/IP. The world wide web, WWW, contains hypertext documents known as web pages. Typical web pages are presented using hypertext markup language (HTML) and accessed using the hypertext transfer protocol (HTTP). A “web address” is technically known as a uniform resource identifier (URI), or more generally, as a uniform resource locator (URL). The URI is an alias for the IP address of the page. Let us consider an example URI, http://www.sethholloway.com. The URI starting with http:// means that the resource we are seeking is available via HTTP. Next comes www, which means that the resource is on the world wide web. The remaining portion, sethholloway.com, is an alias for a specific IP address. Whenever a person tries to access http://www.sethholloway.com using a web browser,
the browser retrieves the HTML information stored at the specific IP address; the HTML is rendered as text and images.

Above, we described the process of retrieving a web page. Here we discuss richer interactions with web pages, namely inputting information. Information on the web is typically transferred using forms. Forms are a generic construct for input; these include text boxes, radio buttons, drop-down menus, and more. To make this idea more concrete, let us consider the ubiquitous search-provider, Google. Google’s minimalist page contains a single input form: a text box form that accepts your search term. Whenever you search, the form input is fed to the web server as a URI. For example, searching for “Seth Holloway” generates the following URI: http://www.google.com/search?hl=en&q=seth+holloway. As with the above example of http://www.sethholloway.com, the Google page is on the world wide web accessible using HTTP. After google.com we see search. Search is the application that is being called. The question mark, ?, signals the beginning of parameters. Parameters are key-value pairs of the form key=value separated by ampersands, (&). In this example there are two parameters, hl=en and q=seth+holloway. The first parameter, hl=en, defines the language, English, abbreviated as en. The second parameter, q=seth+holloway, defines the search query, q, as “seth holloway.” Other web applications will have different parameters, but forms always input information in the aforementioned style, using URIs.

Having described the supporting infrastructure provided by HTTP and HTML, we now move into defining the SEAP architecture.

2.2.2 SEAP Architecture: Devices

As SEAP aims to be generally applicable to a wide variety of pervasive computing applications, a goal of the SEAP architecture is therefore to ensure that the functionality required for these devices is kept to a minimum. To this end, devices are
not required to accept arbitrary inbound connections; instead each device controls its own communication costs through the outbound connections it creates. This includes both the transmission of data requested by an application and the reception of configuration and actuation commands. The SEAP data flow depicted in Figure 2.2 provides a high-level view of the relationships between different hardware components in a SEAP system.

SEAP participating devices are classified as sensors (data producers) or actuators (command consumers). While it is possible for a node to perform both sensing and actuation duties simultaneously, this distinction allows clarity in the architectural description.

**Sensors.** A device that is creating its own data or gathering data from the environment to send to the application is considered a sensor. SEAP enables applications to collect this information from sensors in an intuitive fashion. SEAP, running on the sensor/actuator, creates an interface to the accepted web-based programming approaches. Specifically, when a sensor has data to send, SEAP packages the data and opens an HTTP connection to a preconfigured Uniform Resource Identi-
tifier (URI). The sensor data is encapsulated as parameters in the connection and is transferred to the application. The high-level SEAP algorithm that runs on sensors is shown below.

```java
while(true) {
    connect to data-report-uri
    send readings
    disconnect
    sleep data-report-delay
}
```

The connection to the server is successful when communication paths are available (SEAP delegates communication to an underlying network infrastructure). When the central web server receives the connection and its parameters, SEAP deployments use freely-available, widely-used web application servers (e.g., Apache httpd, Tomcat) to handle communication, parse the parameters, and present the data to the user’s application. Because the user’s application appears to be interacting only through web programming constructs, interaction with sensor data is reduced to operations that are familiar and routine for an average programmer.

**Actuators.** A device that receives commands from an application to alter the logical or physical environment is considered an actuator. As before, SEAP’s goal is to allow applications to pass commands to remote actuators using simple and intuitive techniques. Like sensors, a remote SEAP actuator manages an HTTP connection and invokes native functions. A basic SEAP component for a remote actuator could use the following algorithm.
while(true) {
    connect to command-uri
    while (connection open) {
        read command
        apply command
    }
    sleep command-retry-delay
}

Here, the while (connection open) loop is used to eliminate the resource usage of continuously polling the server for new commands. Devices read sequences of commands from a single connection as they are sent by the server. Assuming the read command above is able to parse the commands as they are delivered, the inner while loop explicitly provides support for batching many commands together. Any sequence of commands returned to the device will be processed in order, without encountering the sleep statement.

However, even batching commands together may still not provide the immediate response that some applications desire. If the connection is closed after each batch of commands is processed, the device must still wait for the command-retry-delay before receiving a new batch of commands. This concern is easily addressed by allowing the read command to block until new commands are received. The server simply distributes commands with arbitrary delays, each of which is read and applied as soon as it is received. When the end-to-end connections are stable (compared to the command-retry-delay), this gives the application much finer control over the timing of actuation commands.

**Reconfiguration.** Both sensors and actuators rely on a small set of variables to properly integrate and support a SEAP application. These variables are set by the programmer, ideally when the device is first placed in the environment. However, while the application is running, it may become necessary to change the values
of such a variable on a remote device. For example, a temperature monitoring application may request temperatures once every five minutes until a threshold is met, then the application will request temperatures once every thirty seconds. To enable these scenarios, devices also host a SEAP component which downloads configuration changes in much the same way that commands are retrieved by an actuator. This allows a developer to programmatically alter any of the variables mentioned in this section, even the reconfiguration URI, at runtime.

In the example sensor and actuator algorithms above, the uri and delay variables can be reassigned by providing new values on the webserver as a simple properties-style configuration page like the one below.

```
http://my.host.name/sensor7/configuration.jsp
configuration-uri=http://my.host.name/sensor7/configuration.jsp
configuration-delay=60
data-report-uri=http://my.host.name/sensor7/temperature.jsp
data-report-delay=3
command-uri=http://my.host.name/sensor7/alarm_status.jsp
command-retry-delay=10
```

In this case the configuration file contains the URIs and delays for a fictitious device “sensor7.” The device parses the configuration and updates its internal variables. Once the sensor has applied this configuration, we expect it to post readings every three seconds, and download its configuration every 60 seconds.

Even devices that can be reconfigured still require initial settings; a process we call bootstrapping. When a device participating in SEAP is initially put into service for an application, the user configures the device by specifying variables necessary to coordinate with the web application. An example configuration utility, Figure 2.3, demonstrates the interaction that is required for this process. Here, a device with a temperature sensor and an LED is configured to publish temperature readings and retrieve commands to control the LED. The nature of this component
depends on the particular target device; its main purpose is to translate between the low-level hardware and protocols and the simple SEAP interface. As the final step of the configuration utility, the SEAP component is loaded on the device (either through a direct connection or through existing over-the-air programming capabilities [29,49,50,59]).

![Configuration Utility: Sensor 0000.0661](image)

Figure 2.3: Example user interface for sensor configuration.

### 2.2.3 SEAP Architecture: Application Server

Because average, hobbyist programmers do not have device-specific knowledge, we shift the application logic to the application server where programmers are more comfortable. By centralizing the logic to an application server we mitigate complexities of coordinating distributed devices. This also allows us to reuse existing high-quality tools and techniques that are not available to similar research projects that develop their own application servers and communication protocols. Because SEAP reduces interactions with both sensors and actuators to a standard web-style interaction paradigm, constructing applications becomes much simpler and more familiar. By relying on standard HTTP, SEAP also allows existing applications to
seamlessly interact with pervasive devices.

A web application framework (e.g., Tomcat, Ruby on Rails, ASP.Net, etc.) simplifies the design of the server by addressing many of the traditional server-programming concerns (e.g., connection management, data-stream parsing, etc.). Many of these frameworks also provide support for advanced features such as load balancing and clustering that would be useful to future large-scale pervasive computing applications; these features, enabled specifically by our approach and not present in other research, would allow pervasive computing to scale using standard methods. While other approaches would necessitate research into optimization and scaling, SEAP applications can grow and evolve using the readily-available features. By including web application frameworks in the SEAP architecture, we are encouraging the separation of non-application concerns from the user’s code base and into a purpose-built tool. Pragmatically, the key advantages of using a web application framework are derived from re-framing misunderstood pervasive computing problems into standard web applications. This enables developers to use almost any of the popular programming languages and the manuals, tutorials, and guides that are available for them. Developers also benefit from the high-quality tools available for testing, debugging, documentation, and integration for this popular application domain.

By introducing elements that serve as both device and server, applications can adopt a more hierarchical design. Intermediate nodes could perform data filtering, aggregation, or even pre-processing.

While we have primarily provided details about dynamic web pages, SEAP also works with simple web servers with static content. Commands and configurations requested by remote devices could be read directly from the central server’s file system. Any web server would be sufficient for this purpose. Updates to the files made by hand or by other applications would be immediately reflected by the
2.3 SEAP-Based Implementations

We are interested in more than theory, so we must further develop the theoretical model described above. We address this with our second research task: **RT2. Create a working embodiment of the sensor/actuator model as a middleware, SEAP (Sensor Enablement for the Average Programmer).**

SEAP eases software development by moving the logic of pervasive computing applications to an environment that is accessible and familiar to average programmers. In this section we detail how this is done in practice for two different applications. One application was developed before the SEAP architecture was conceived and then adapted to it after the fact. The other was developed from scratch using the SEAP patterns to build a robust, flexible, and easily testable application. We mention the testability because average programmers are less skilled at debugging than their senior counterparts [45]; thus straightforward verification is critical to their success.

2.3.1 SEAP Applied to an Existing Application

As graduate students, we consume large quantities of caffeine—most commonly coffee. Long hours writing text and code require a steady supply of black gold, but brewing two cups at a time is wasteful and inefficient. To better organize the coffee supply, students formed an unofficial coffee cooperative. At first, the student who brewed coffee would send an email to others in the group. This solution was rather cumbersome so an industrious graduate student developed a web application, *UbiCoffee*, to monitor coffee levels at several locations in our building. Originally, users of the application updated the “amount available” form through a web-based interface; the application logic would simply display the previously input value.
Since a web browser was often not available near the coffee pot, most users traveled back to their desk to update the application. Often these trips were interrupted, leaving the application unaware of the actual state of the coffee pots.

To increase the quality of the data reported to the application, we wanted to place two buttons near the coffee pots to indicate the addition and removal of coffee. Instead of directly inputting the number of cups of coffee available, users would simply input the change. Users would simply press a button labeled “+1” several times when brewing new coffee (8 presses to indicate brewing 8 cups) or press the “-1” button when taking a cup from the pot. To provide this functionality, we selected SunSPOT [66] devices. However, this choice left us with a problem. The web application is written with Ruby on Rails [7], while the SunSPOT is programmed with a variant of JavaME [55], two languages that do not interoperate natively.

SEAP works perfectly in this situation by providing a defined interface for communicating using HTTP. To link the SunSPOT application to our UbiCoffee application, both must communicate using HTTP. Being a web language, Ruby on Rails does this out of the box. Integrating the SunSPOT with HTTP was more involved because the SunSPOT physical layer is a Zigbee-like protocol; SunSPOTS do not support typical Internet protocols like 802.3 (Ethernet) or 802.11 (wireless local area networking). Using JavaME, we developed an application that created an HTTP stream that was sent to a basestation SunSPOT (connected to a standard desktop personal computer). The basestation forwards the HTTP packets using the host machine’s network. This task was not overly difficult; however, in the future, device manufacturers could easily provide this functionality out-of-the-box.

With SunSPOTs now able to communicate with the web server, we could replace the user input with SunSPOT inputs. To start, we extended the UbiCoffee application to include an “amount changed” form in addition to the existing “amount available” form. This work was completed quickly by the original devel-
opers and could be independently tested and verified without any interaction with the SunSPOT. Figure 2.4 shows the data flow for UbiCoffee before and after SEAP. Because the SEAP version of the application can interact with users and devices, it is more powerful than the standard, device-free application. We gain this power while the logic remains virtually unchanged—exactly what “average programmers” need.

![Figure 2.4: Data flow for UbiCoffee, before and after SEAP.](image)

The second step was programming the SunSPOT to behave as a simple pervasive computing device with “+1” and “-1” buttons. Now, when the buttons are pressed, the device posts the appropriate value to a given web URI as a given parameter. To link our new pervasive device to our web application, we simply provided the URI of the new UbiCoffee page and the parameter name to the device. Concretely, this means that the SunSPOT would send an HTTP packet to `http://192.168.0.1/UbiCoffee?amountChanged=1`. The resulting system now accepts data from users via the SunSPOT device or the traditional web interface.
2.3.2 SEAP From Scratch

As part of on-going research into a pervasive computing testbed (PCTB) [6], we have access to a wide variety of robotic components like remote-control vehicles, pan-and-tilt servos, cameras, infrared sensors, sound sensors, GPS systems, and passive-and active-RFID equipment. Ultimately, the PCTB would like the components to interconnect and allow the vehicles to be controlled programmatically. We decided that SEAP was an ideal method to provide this functionality, and Spot-to-Bot was born. Shown in Figure 2.5, the Spot-to-Bot application [69] was developed by an undergraduate researcher in our lab to steer a Roomba robot using the accelerometer on a SunSPOT. In this case, the SEAP architecture was used to break the complete application into three distinct parts, each independently testable and exchangeable.

![Figure 2.5: The Seap-to-Bot data flow.](image)

The first of the three components is a web application with three pages. The first page is a web form that accepts three values: “X”, “Y”, and “Z”. Eventually these values were to be provided by the SunSPOT accelerometer. In the meantime, these values were provided by submitting the form from a web-browser. The second and third pages of the web application display one value each: the requested speed and turning angle for the robot. The web application thus contains the logic to translate the raw data provided by the sensor into the commands given to the actuator.

The second component of the system delivers accelerometer data from the
SunSPOT device to the web server. This component is a simple extension of the program developed for the *UbiCoffee* application described in the previous section. The third system component drives the Roomba robot. Using existing code as a basis, the majority of this work fell to retrieving and parsing values provided by the web server.

The resulting application uses three components that can be independently tested and verified. We are able to easily test each component because we use standard protocols; our model for sensors and actuators takes the place of standard web forms, so users can debug using familiar web forms rather than obscure pervasive computing devices. The resulting application is flexible and easily extended to accept new sensors, actuators, or behaviors. In fact, the three components can even use different programming languages, each specifically suited to the component’s primary task (e.g., JavaME for the SunSPOT, PHP for the server, C++ for the Roomba controller).

### 2.4 SEAP Evaluation

We have created several simple applications using SEAP, but, as the creators, our experience is less informative than the experiences of our intended audience: average programmers. To evaluate SEAP, we will perform a user study and complete the final research task for this work: **RT3. Evaluate the usefulness of SEAP on a selection of novice software developers.**

We will recruit 15 “average programmers” for a user study that would mirror the SEAP development process if they were in the wild. We plan to recruit these students with a letter targeted at Computer Science and Electrical Engineering undergraduate students.

For maximum efficiency, we will test multiple students simultaneously. Students will be tested in a classroom setting in groups of five. We will begin with a
pre-test to assess general competency. This pre-test will test students on

- algorithmic design of an application they will be asked to write;
- embedded programming using strict C programming;
- and web programming using a language of their choice such as PHP, Perl, Python, or Java.

Together, these three tasks will provide insight into students’ (representative of “average programmers”) abilities. We hypothesize that students will understand the basic logic, struggle with embedded programming, and perform moderately well with web programming. After the pre-test we will then provide a brief tutorial on the SEAP system and background information on the necessary technologies as a short lecture.

The test will require users to create and alter applications using SEAP. Participants will have all materials provided (computer, sensors, actuators, and printed instructions) and will be allowed to ask the test administrators for general help. (All questions will be documented as part of the study.) Being an involved, multi-part test, we will compensate users.

2.4.1 Study Tasks

There are two high-level actions for programmers using SEAP: alter an existing application (described in Section 2.3.1) or develop an application from scratch (as in Section 2.3.2). We wish to test the basic SEAP actions with the following three tasks:

- Alter an existing SunSPOT application to use a different sensor as input. We will provide the Seap-to-Bot application (described in Section 2.3.2) and ask users to move the robot with a different device.
• Alter an existing web application to replace user input with SunSPOT input. We will provide access to the UbiCoffee application (Section 2.3.1) and SunSPOTs then ask users to edit UbiCoffee to accept input from a SunSPOT instead of direct webpage manipulation.

• Develop a new application given the specifications and all necessary hardware/software.

2.4.2 Questions We Hope to Answer

We are primarily interested in assessing the ease-of-use of SEAP. If users cannot complete the tasks, the problem may be education, not SEAP, so we plan to tease apart these two issues using qualitative and quantitative data. Users will provide valuable qualitative feedback when asked:

• How easy was the framework to use?

• Would improved documentation/support make the system easier to use?

We expect that users will be able to use the system without a deeper understanding of sensors or the Internet; however, we will not rely solely on their opinions. We will verify these answers using objective measures. For each study participant we will collect data for the following questions:

• Did the participant complete the test?

• How long did it take the participant to complete the test?

• How many questions did the participant ask?

• What questions did the participant ask?

Taken singly, user replies will not be particularly illuminating; however, the aggregate data will help us explore these meta-ideas:
• Were there any common questions asked?
• Was there a step that slowed a majority of people?
• Is the framework usable in its current form?
• If users struggled, would improved documentation remedy the problem?

2.5 Remaining Tasks

Work on SEAP provides three major research contributions: a method for modeling embedded devices as web-compatible objects including a mechanism for dynamic configuration, the embodiment of these concepts as a middleware, and a user study that proves that SEAP can be understood and used by undergraduates. We have worked extensively with SEAP, developing several applications and exploring the possibilities of the middleware. We have developed applications from scratch and altered existing web applications, allowing them to interact with sensors and actuators dynamically. We have also demonstrated and presented the research to great fanfare. We are now ready to allow a subset of our target audience to use the SEAP middleware. We plan to test outsiders as described in Section 2.4.

2.6 The Potential Power of SEAP

From cellular phones to motions sensors and smoke alarms, sensors and actuators are all around us. The next evolution in computing will connect sensors and actuators with standard computers and enable a host of formerly futuristic pervasive computing applications. Currently, only researchers and highly trained professionals can program embedded devices, while “average programmers” pursue high-level languages like Java. We must bridge the gap between highly specialized device programmers and “average programmers” so the massive amounts of necessary code
can be developed quickly. Sensor enablement for the average programmer, SEAP, allows “average programmers” to develop meaningful, custom applications without learning the low-level details of sensors and sensor networks.

While we could have developed a new, heavyweight framework aimed at making ubiquitous computing more accessible, we instead focused on a broad solution using existing technologies where possible. As a result, the SEAP middleware allows people to interact with sensors and actuators without learning new languages or procedures. The scalability and accessibility of the Internet make the web an ideal platform for increasing the number of active pervasive computing applications. Initially, only large organizations had a web presence. However, the Internet age truly burgeoned once individuals could create and maintain web pages. To achieve the same growth in pervasive computing, sensors and actuators should be easily accessible through a suite of abstractions usable by the average programmer. The SEAP middleware achieves this, allowing programmers to easily integrate sensors and actuators without any awareness of the specific low-level languages and protocols.

SEAP takes advantage of the vast body of work on web programming to provide an approach that is easy-to-understand, easy-to-use, easy-to-test, language-agnostic, robust, reusable, and immediately achievable. People are ready for pervasive computing applications; we provide an accessible method to enable multi-device developments. SEAP is that method.
Chapter 3

I SEEU—the World Should Too

Sensor Enablement for End-Users

Existing sensor network application deployments vary widely in several dimensions, including implementation language, employed communication protocols, handled resource constraints, etc. SEAP is effective in easing these burdens for average programmers, but because SEAP requires users to understand a programming language, it is inadequate for people with little or no programming experience and a low tolerance for tinkering. The future of intelligent environments is pervasive, from smart homes to intelligent offices and everywhere in between. It is imperative that everyone be involved in intelligent environment deployments for two reasons: intelligent environments are most powerful when customized by the people that will use them and there are simply too many installations for professionals to perform in a timely and cost-effective fashion.

To demonstrate the need for customization, let us consider an intelligent construction site—a specific class of intelligent environment that endeavors to improve
the construction process. An intelligent construction site is one that has been instrumented with sensors and actuators that will allow multiple unrelated events to be monitored; for example, cranes would be outfitted with accelerometers and GPS that work together to provide a warning when two cranes are in danger of colliding. On a large project there are many workers, tons of building materials, hundreds of tools, numerous large machines, as well as permits and regulations that must be tracked. Experts in construction recognize that intelligence derived from the coordination of many devices on-site would greatly improve productivity and safety [30]. Materials, machinery, people, and topology differ widely between projects, so an intelligent construction site only makes sense when the software improves the conditions (safety, productivity, cost, etc.) for the specific site. Currently, building intelligent construction sites requires additional people because the domain experts (e.g., construction site supervisors) cannot develop or tailor applications to their unique requirements. The power of the intelligent construction site springs from customizing the software; who better to customize the environment than domain experts, the people on-site who know the specifics of the project?

To further demonstrate the need to empower everyone, let us consider smart homes—another class of intelligent environment that increase quality of life at home. Smart homes allow users to automate actions to increase comfort, lower costs, and reduce waste. Common smart home applications automatically adjust the temperature and light levels as well as audio and video settings. We must involve end-users, in this case the inhabitants of the smart home, because the time and money necessary for an expert to set up an intelligent environment is large and will not scale with the number of potential installation locations. Given the glut of potential installation locations\(^1\), it is clear that intelligent environments will not reach the mainstream without the direct involvement of end-users in their definition and deployment.

\(^1\)In the year 2000, there were 105,480,101 households in the United States of America [12].
Our research will allow domain experts to easily customize their environment without the overhead of current intelligent environment deployments, overhead such as obscure programming languages and communication protocols and complex interfaces. We will empower end-users (people with no formal training or previous experience with the technology) to specify behaviors in their own personalized applications as we address the research question, “How can we enable end-users to specify behaviors and create intelligent environments?” There are four research tasks associated with answering this research question:

- **RT4.** Discover how end-users think about a tangible intelligent environment, smart homes, and how they express desired computer-controlled home behaviors.

- **RT5.** Design a programming interface that is accessible to end-users for creating customized intelligent environment behaviors.

- **RT6.** Create a software system, SEEU, that allows end-users to specify behaviors without in-depth knowledge about the underlying technologies.

- **RT7.** Evaluate the effectiveness of SEEU on end-users.

To spread intelligent environments we must open the development and make it easy for people with no formal training to customize their environments. An end-user programming framework that is not only extensible and flexible but also fits the mental model of a majority of people satisfies the requirements. This end-user programming framework must support the development of completely new systems as well as integrate existing infrastructure and applications. Along the same lines, the framework must allow intelligent environments to be deployed as complete solutions or to be built piecemeal. To further complicate this task, the framework must be built to function on sensors and actuators, widely varying resource-constrained devices with relatively few onboard capabilities [13,23,61]. An intelligent environment
deployment system is challenging because of obscure hardware, necessarily simple software, and the novel human-computer interactions.

Researchers have attempted to spread intelligent environments and empower end-users in a number of domains; we present several notable projects in Section 3.1. To overcome existing problems and provide a novel, useful system, we plan to involve users throughout the project as described in Section 3.2. Specifically, we plan to survey users and discover how they would want to “program” an aware home (Section 3.2.1). With this information we will create a study utilizing paper prototypes covered in Section 3.2.2. Section 3.2.3 details the development of a software system that incorporates our previous results. With the system in place, we will evaluate our overall success with a user study as described in Section 3.3. We cover remaining tasks and impact in Section 3.4 and Section 3.5, respectively.

3.1 The Latest in a Long Line

Most recent research into intelligent environments focuses largely on smart homes. Within this context there are two broad categories: industrial and academic. Because of their immediate usefulness, intelligent environments receive commercial attention; however, innumerable questions and visions still necessitate research. Researchers are studying societal issues with intelligent environments, how to extend healthcare to the smart home, which interfaces end-users prefer, and myriad technological issues.

There are many companies that offer hardware and software for intelligent environments, including Control4 [1], Home Automation, Inc. [3], Creston [2], Universal Devices [11], and Teletrol [10]. There are scads of products that add intelligence to home audio and video distribution such as Sonos [9], Niles [5], SelcTTouch [8], and the Logitech Harmony remote [48]. These consumer products offer individualized, closed solutions that cannot be easily customized, nor can components in these
systems be quickly replaced or upgraded. End-users must be able to develop their own intelligent environments using any and all hardware so they can adapt their intelligent environment quickly and easily. One of the most widely used product lines is based on the X10 protocol [70], which controls devices over the power lines in a building. X10 components are relatively inexpensive and widely available, but, given the lack of a graphical user interface (GUI), creating an X10 system requires users to read and understand electronic diagrams and control systems.

While consumer products have several problems, there is hope for the future as current problems are being solved in academia. Research into intelligent environments has produced two major approaches: machine learning [33] and end-user programming [56]. Machine learning leverages recent advances in hardware and algorithms to automatically learn users’ behaviors. End-user programming provides an explicit method for a user to describe the desired intelligence.

Environments using machine learning techniques eliminate the users’ need to specify behaviors. These approaches make sense for some complex situations, like home healthcare, but are inadequate for an entire residence. There must be a human in the loop in case the machine learns improperly; for example, consider a couple with a newborn baby. For the first six months, their routine is sporadic with frequent feedings and changings multiplexed on a normal adult routine. Even an advanced machine learning system would incorrectly identify 2am as an “awake” time. Early activity recognition schemes [71] were imprecise, with prediction accuracies as low as 25%. Later approaches, such as the MavHome [25], have improved accuracy to at least 47%. However, the MavHome system requires preset information to monitor for associations; for example, the system correlates events based on a pool of events like AlarmOn, AlarmOff, BedroomLightOn, BedroomLightOff, etc. Intelligent environments are open-ended and extensible so a priori knowledge cannot handle all future configurations. CASAS [58] provides a framework to au-
tomatically learn behaviors and also provides the user with basic feedback abilities. CASAS is a good first step, but the system cannot provide the full range of expressiveness necessary for end-users to automate their homes; CASAS is inadequate for tasks that cannot be learned. Machine learning approaches that provide services like positioning can be very useful; for example, PowerLine Positioning [57] uses existing power infrastructure to locate inhabitants based on electrical signals. Machine learning techniques like this cannot create an entire smart home, but they could be used as abstractions for larger smart home implementations.

On the other end of the spectrum, end-user programming allows users to explicitly specify components and their connections. End-user programming can be accomplished through a wide variety of paradigms, though the most effective by human computer interaction principles [26] are human-centric and only moderately abstract [46]. CAMP [73], created a “magnetic poetry” interface that mimics kitschy refrigerator magnets popular around the turn of the millennium. These magnets adorned the cool kitchen appliance with a subset of words that industrious participants would rearrange into phrases. CAMP provided a novel new interface for automated capture and playback, a subset of smart home tasks. The interface was developed to fit the mental model revealed by a user study [74]. Another project, Media Cubes [16] offers a tangible programming interface [42] similar to remote controls. With Media Cubes, inhabitants associate each side of the cube with a specific device’s action; for example, turning on the television. Users can later replay the action by turning the cube to the desired cube face. The CRISTAL system [35] offers an end-user programming interface based on the tabletop integrated computer [4,37,53]. The tabletop displays a live image from a camera mounted overhead; the system is programmed to provide cues for interacting with each device, such as a play button superimposed on the television. This is an interesting idea, but there are many privacy concerns in displaying live images from an environment.
In “Playing with the Bits” [41], users snap available components together using a jig-saw puzzle metaphor. Unfortunately, the subset of devices (in this metaphor, pieces) is limited by what the researchers can produce, so the system is not easily extended. User-centric techniques hold great promise for allowing users to create their own personalized smart homes.

3.2 User Driven Design: The SEEU Approach

As described previously, there are numerous frameworks and interfaces that were created with little or no consideration of end-users. We propose a principled approach as we design the SEEU system; the basis of which is end-users. As such, we plan to guide our design decisions using input from end-users—beginning before design. To properly provide an effective interface for specifying behaviors in intelligent environments, we must first determine how users think about intelligent environments. We will gather data in a simple online survey. After analyzing this data, we will combine the results with our creativity and knowledge of existing approaches to develop several prototype interfaces matched to our survey results. These paper prototypes will then be subject to another round of end-user testing. In this study, people will help us determine which style of interface best supports their mental model and balances the needed expressiveness with ease-of-use. Armed with an effective interface, we will create the software system. Finally, we will solicit feedback about the SEEU software system from end-users.

3.2.1 Learning How Users Think... About Intelligent Environments

In this section, we begin to answer our research question by tackling the first of our four research tasks: **RT4. Discover how end-users think about a tangible intelligent environment, smart homes, and how they express desired computer-controlled home**
While a one-on-one session would have allowed more flexibility and encouraged longer responses, a survey was ideal for quickly learning how a large number of end-users think about smart homes. We surveyed a random sample of end-users about their notions of intelligent environments, specifically how they would “program” a smart home. We chose the smart home domain because we felt it was more familiar and approachable to people outside the field of pervasive computing and software engineering. In the survey we hope to learn how and what users think about smart homes. We are particularly interested in opinions about home automation, what abstractions end-users would find useful to customizing behavior, and what devices and capabilities are expected or desired. Armed with this data, we can synthesize the information into a mental model of end-users for intelligent environments and create a matching programming interface.

**Design of the Survey**

In this first phase, we were interested in gathering as many responses as we could, so we needed an empirical method that scales easily. This makes a web accessible survey—an ideal choice. The only condition for participating in the survey was adult status—children are likely to use smart homes too, but they are not the immediate aim of this research.

Previously, CAMP [73] found differences in how parents versus non-parents described aware home interactions; specifically, parents considered the entire family, while single people were more self-centered in their thinking. For this reason, we asked participants their familial status in hopes of verifying these conclusions. There are also age and sex stereotypes about homes and technology, so we asked participants for these pieces of information. The final piece of information we are

---

2 The survey materials can be found in the Appendix A.
interested in is technological expertise. We believe that technologically savvy people will be the first to adopt smart home technologies, however, we are interested in making a system that fits everyone so we asked the survey participants to self-assess their degree of technological expertise. We designed the survey to take roughly 30 minutes.

To disseminate the study we invited participation from roughly 25 colleagues, friends, and family with some interest in intelligent environments or user studies. In the call for participation we invited our recipients to freely forward the survey. The survey was available online for ten days; we hoped to receive at least 30 valid responses to establish a baseline of statistical significance.

Our questions were designed to provide examples of smart home policies and allow users to create their own unique policies. To set the mindframe of the participants, the survey started simply, fixing the action taken by a smart home and asking participants to explain the conditions upon which they would want that event triggered in their smart home. For this question, we asked the participants how they would like the heating and cooling system of their house controlled given complete control; i.e., what sensed information should impact changing the house’s temperature in a room. We gave the participants a sample list of sensors that could be used for this purpose, including temperature sensors widely distributed in the home, sensors for tracking the locations of the home’s inhabitants, etc. Next, we asked users to describe what sensed events would satisfy the aggregate condition of burglary. With home security in mind, we then asked users what actuators would be activated on the condition of a burglary. Finally, we asked participants to describe any other policy they desire from a smart home. This final question was meant to evoke lengthy, unique responses about exciting personalized combinations of sensors and actuators.
What Will Users Say and How Will They Say It?

The survey itself was designed to help answer the following questions.

- Did users describe the sensing task or the actuating task first?
- Were there any specific words or sentence constructs used by all or most of the respondents?
- Were there common applications described?
- Researchers have posed a variety of smart home capabilities; did average users think of any of these situations?
- Did users have any concerns about intelligent environments?
- Are there noticeable trends based on age, sex, technological expertise, or familial status?

Users Chime In: Survey Results

In 10 days online we received a great deal of interest in the survey; ultimately, there were 77 surveys started and 64 completed. Participants ranged in age from 19 to 85 with an average age of 41.7 years. The respondent’s locations are displayed on the map in Figure 3.1; we had responses from Hawaii to the Netherlands, with the bulk scattered across the United States. The average self-assessed technological expertise rating was a little higher than medium: 2.32 out of 3 (1 being novice and 3 being expert). Of the completed surveys, 28 respondents were male and the remaining 36 were female. A little over half the participants, 35, were parents. There were no fundamental differences in the responses between the sexes, nor did familial status impart a discernible difference.

We found that 10 participants had ecological concerns—not surprising given the recent rise of the green movement. These participants used phrases like “energy
efficiency” and “waste” as well as judgement words (“bad,” “good,” etc.) with respect to electricity usage; these responses were likely to include mention of the term “environment” meaning Earth.

A majority of respondents, 73.44%, used some form of conditional logic (“if,” “then,” “else,” “while,” “when,” “otherwise”) to describe the smart home scenarios. This way of reasoning was expected from people with high technical competence (for example, programmers who are used to coding with similar reserved words), but the use of conditional logic was surprisingly common across all demographics.

Another discernible trend presented itself when 22 responses specifically mentioned the ability to control and monitor their home from anywhere. We believe most people would want this ability, but, because the option was not explicit, respondents did not know to ask for it.

The next largest technological implication was the request for some form of machine learning. Several participants, 25%, expected the home to have implicit knowledge about their schedule or the ability to learn their schedule.

Very few users detailed situations described in research such as Mark Weiser’s classic example of Sal’s routine [77] in which her home automatically brewed coffee as she rose from bed, her car guided her to a shorter route, and monitors in her proximity automatically displayed messages for her. Rather than repeat well-known research scenarios, it was more likely that users described functionality that is al-
ready commercially available, albeit obscure, expensive, and proprietary. We believe this demonstrates just how ready people are for a cohesive system that allows them to encode basic actions. These results in particular demonstrate that a system that optimizes all possible technological issues or creates a radical new interface to satisfy all possible situations may not be the right way to spread intelligent environments. Instead, we need to ease into supporting end-user programming of smart homes in a way that is extensible so that end-users will eventually be able to create applications that they cannot yet conceptualize.

This survey collected responses from an intelligent environment that users may have some basic understanding of and one example that people can relate to, the smart home. The questions were relatively open-ended, which led to varied responses; the responses revealed several trends that we believe will exist in most if not all intelligent environments.

3.2.2 Developing an Effective Interface

A number of previous approaches have been device- or technology-centric, which generally leads to incredibly powerful systems that are hard for people to use. Instead, we take a user-centric approach that centers on an effective interface, leading us to our second research task: **RT5. Design a programming interface for intelligent environments that is usable by end-users.**

Based on the survey, we feel the standard personal computer (with mouse and keyboard) is an ideal platform for interacting with or programming behaviors for smart homes. No one complained about the format of the survey or the method of interactions; our users were all comfortable interacting with a web browser, typing and clicking. Because a majority of participants expressed their answers using conditional logic, an interface that allows users to express home behaviors in conditional logic would fit the existing mental model of users; most responses were in
the form \textbf{IF CONDITION THEN ACTION}. Comfort with the online survey and a strong
desire for the ability to interact with their smart home remotely lead us to believe
the system should be accessible online. This prescribed functionality is largely de-
coupled from the interface, so this phase focuses on determining a proper front-end
that will enable end-user programming of intelligent environments.

\textbf{Testing Prototypes On Average People}

There are numerous potential computer interfaces to consider for intelligent environ-
ments. The primary purpose of the interface is allowing the user to specify behaviors
for the space; the interfaces will differ in how they represent devices and how users
input their desired functionality. Developing a high-fidelity graphical user inter-
face takes significant time and provides no significant benefit over low-fidelity paper
prototypes \cite{75}, so we will evaluate the core interactions using paper prototypes
that represent what the user would see on screen. Beyond assessing usability, paper
prototypes can also expose missing requirements, reveal preference in one design
amongst alternatives, focus priorities, and reveal issues beyond the interface \cite{67}.

In a traditional programming interface, a developer would interact directly
with devices through code—probably an obscure language like NesC \cite{32} that works
only on a small subset of specialized embedded hardware—and would be respon-
sible for creating his own methods for accessing the data and issuing commands.
This interface is slow even for seasoned developers and it absolutely will not work
for end-users. An effective interface will allow end-users to leverage the capabilities
of devices without explicit knowledge of the specific devices. To accomplish this,
the interface should abstract the capabilities to plain language; for example, “tem-
perature” rather than “EDemoBoard.getInstance().getADCTemperature().” After
defining a set of basic abstractions there are still questions as to how to present the
options to a user. Accessing these abstractions could be accomplished in a variety
of ways, and we want to discover an effective interface.

We would like to test users using three prototypes that vary with respect to flexibility and simplicity:

- **Prototype A**: A puzzle-like interface in which users choose pieces that fit together

- **Prototype B**: A magnetic poetry type interface that allows users to freely specify behaviors by creating sentences given a set of words

- **Prototype C**: An interface that imposes a conditional logic framework in which users select the conditions and events from a list

The puzzle interface will show “puzzle pieces” representing sensors and actuators. To connect sensing and actuation, users will have to set the parameters for the devices. We will also consider the use of natural language input versus a fill-in-the-blank “mad-libs” style interface similar to magnetic poetry. The last interface, the conditional logic framework, will have users select conditions (corresponding to capabilities of sensors) and actions (provided by actuators) from a menu.

Taking into account the time necessary to administer the test and analyze results, we plan to test 12 subjects. To avoid ordering bias we will vary the interfaces, and we will run two participants each for the six potential orderings (A-B-C, A-C-B, B-A-C, B-C-A, C-A-B, and C-B-A). Participants will be compensated for their time and will be selected on a first-come, first-served basis after an open recruitment letter is sent.

The test will take roughly one hour. We will begin by collecting paperwork then explain the experiment. To engage the users and prevent “writer's block,” we will begin by explaining the smart home concept including a list of example smart home policies. After the instructions we will begin the test. During that time, users will be asked to specify two behaviors using each interface: first, a policy to
control the thermostat, then a policy to detect an intruder. We will present the paper prototype then ask the user to perform each activity, in sequence, with the prototype. We will encourage users to work through the task on their own while verbalizing their thought process. To capture the data we will film the sessions. We will measure a user’s level of confusion with the number of times they request assistance. To get a rough idea of which interface is most intuitive we will also record time-on-task; despite being skewed by the “think aloud” method, we believe time-on-task will reveal some differences.

What We Hope to Learn About the Interfaces

As we perform the study we are interested in answering several fundamental questions.

- Is natural language input more effective than a specified framework?
- Can users understand the function definitions (including parameters) from the underlying hardware?

Beyond gathering quantitative data about time-on-task and the requests for assistance, we are interested in users’ qualitative feedback. The qualitative feedback will be collected and analyzed similar to the survey results (Section 3.2.1). We hope that the interface that users like the most will be the interface that facilitated the quickest completions with the fewest number of questions. In the course of our experiments are open to finding a hybrid interface that better suits the users.

3.2.3 Putting it all together

After two rounds of user-centered tests, we will have a good idea of the types of tasks users want to perform and what types of interfaces best suit these tasks. This brings us to the concrete embodiment of the previous research: RT6. Create the
software system, SEEU, that allows end-users to specify behaviors without in-depth knowledge about the underlying technologies.

The SEEU system will provide an expressive end-user development system for intelligent environments.

Mr. Holloway’s Opus: The SEEU System Architecture

Because of its flexibility and expressive power, we plan to employ the SEAP system as the basis for SEEU. This is not surprising since both systems aim to reduce the barriers to entry to intelligent environments. SEAP simplifies the development of intelligent environments by creating programming constructs that novice programmers can understand. SEAP solves many of the hardware difficulties inherent in working with resource-constrained devices, but offers no usable interface such as is required by end-users. SEEU is largely a human-computer interaction contribution that makes SEAP accessible not only to programmers, but to everyone.

SEAP will be adapted into the larger SEEU framework, although the specifics of how SEAP will change will largely depend on the interface chosen. The SEEU system requires the ability to automatically program and reprogram devices. This task requires linking the front-end, the interface, to the back-end, SEAP. The user’s few simple mouse clicks will actually require hundreds of lines of code involving multiple devices, a database, and a web server.

Our final product will be a computer-based end-user development application accessible via the Internet. The demonstration application will allow users to specify behaviors in an aware home given a small set of devices. We plan to employ SunSPOT devices because we are familiar with the technology and have already proven SEAP using these devices; this limits us to the capabilities of a SunSPOT: three-dimensional accelerometer, two push buttons, a light sensor, a temperature sensor, and eight LEDs. Although we will develop the demonstration
using SunSPOTs, the system would also work for other devices. SEEU is a general purpose solution that we will extend as part of future work.

### 3.3 Can End-Users SEEU?

Once the SEEU system has been built, we must evaluate our success at enabling end-users to specify behaviors in an intelligent environment—our fourth research task: RT7. *Evaluate the effectiveness of SEEU on a subset of end-users.*

**Studying Ourselves**

The largest, and arguably most important user study of this work is the usability test of our software system.

Once the system has been built, we will evaluate its usability on at least 10 average adults. The test will be administered in the Mobile and Pervasive Computing (MPC) group’s intelligent environment room. The study will take approximately one hour, and participants will be compensated for their time.

The test will consist of two tasks: one to create a policy we describe and one to develop a behavior of the participant’s choosing. Success of the interface will be based on the users' abilities to successfully complete the tasks and their qualitative feedback about using the interface.

**We Have Questions; Users Have Answers**

- Were users able to complete the task in a timely manner?
- Did people like the system?

Based on the results, we will provide recommendations for future research and offer suggestions to increase the prevalence of intelligent environments.
3.4 Remaining Tasks

Work on SEEU provides three major research contributions: the mental model an end-user has when thinking about intelligent environments, a graphical interface to SEAP that allows users to intuitively specify behaviors for intelligent environments, and a software system that can provide the technical foundation necessary to support end-users and the limitless number of current and future devices.

The majority of the work in this chapter is currently incomplete. The survey has been designed and disseminated, and data from the survey has been collected. While we have some idea how to do the paper prototype and the software system studies, the report and IRB approval are incomplete. The idea for the software system is emerging, but is completely informal and unwritten.

3.5 What Happens When We All SEEU

Intelligent environments will not flourish unless end-users are able to tailor the system to their liking! The technology necessary to create intelligent environments (sensors and actuators, computers, wireless networking, etc.) is or will be available within the next five years; however, no cohesive framework nor usable interface exists. Sensor enablement for end-users, SEEU, plugs this hole by providing a usable framework for specifying behaviors in intelligent environments.
Chapter 4

Research Timeline

The expected timeline for completion of the proposed work, the dissertation document, and defense is shown in Figure 4.1. I expect the work to be completed by December 2010.

![Figure 4.1: Research Timeline](image)

Figure 4.1: Research Timeline
Chapter 5

Conclusion

In this proposal, I introduced the research steps necessary to create an end-user programming framework for intelligent environments. I first provided the theoretical foundations to transform sensors to a more well-understood paradigm. Building on this foundation, I presented a specification technique that allows users to quickly use and configure pervasive devices. I then demonstrated the technique in a data transportation middleware, SEAP, that empowers even novice programmers to create pervasive computing applications. Taking SEAP one step further, I further reduced the complexities of sensors and actuators with an expressive, easy to use interface. This interface has been empirically studied and provides a solid base for future development. The ultimate goal of this research is to lower the barrier for entry into creating pervasive computing applications. This research thus facilitates the adoption of pervasive applications fostering the third wave of computing.
Appendix A

Appendix

A.1 SEEU Smart Home Survey

This section describes the SEEU smart home survey detailed in Section 3.2.1

A.1.1 Recruitment Letter

Professor Christine Julien and PhD student Seth Holloway from The University of Texas at Austin are studying smart homes and we want to hear from you! Anyone over the age of 18 is encouraged to participate.

The purpose of this study is to discover how the general populace conceives of smart homes. We hope to gather a baseline level for research in a fundamentally new approach to interacting with smart homes. The goals of this research are to support the development of intuitive interfaces for everyone with the hope of providing vastly more efficient and effective means of interacting with the home. Your assistance will help us advance this exciting new research area!

https://www.surveymonkey.com/s.aspx?sm=sHs4AQr15nT5jnRzmhJiwA_3d_3d

We expect each study to take approximately 30 minutes. Anyone over the age of 18 is encouraged to participate. Your opinion is highly valued and greatly
appreciated!

Thanks,

Christine Julien and Seth Holloway

A.1.2 Cover Letter

You are invited to participate in the Survey on Smart Homes. This study is being conducted by Professor Christine Julien of the Department of Electrical and Computer Engineering at The University of Texas at Austin (512-232-5671, c.julien@mail.utexas.edu).

This purpose of this study is to discover how the general populace conceives of smart homes. We hope to gather a baseline level to drive research in a fundamentally new approach to interacting with smart homes. The goals of this research are to support the development of intuitive interfaces for defining behavior of a smart home with the hope of providing vastly more efficient and effective means of interacting with the home. Your assistance will help us advance this exciting new research area.

We seek to establish the base levels of knowledge about smart homes and how users conceive of interacting with a smart home given current technologies. There are no right or wrong answers! We currently know very little about the subject so testing human subjects is vital in developing the future of smart homes. We are not testing an individual’s knowledge and abilities; answers will be used to establish the proper abstractions and parameters to provide in home automation.

In this study, the only information we collect from you is about your life situations. Any identifying information, such as your email address if you would like to participate in future studies, will be separated. Your data will be stored with a unique participant identifier to ensure that we do not inadvertently use a participant twice (thereby tainting the internal validity of the studies with learning effects).

We expect each study to take approximately 30 minutes.
There is no risk associated with this study. There will be no costs for participation. Your participation is voluntary, and you have the right to refuse to participate or withdraw at anytime during the study without penalty. We will take all possible measures to insure your privacy and the confidentiality of the test results. Your participation in the study provides your implicit consent to participate in this study.

This study has been reviewed and approved by The University of Texas at Austin Institutional Review Board. If you have questions about your rights as a study participant, or are dissatisfied at any time with any aspect of this study, you may contact - anonymously, if you wish - the Institutional Review Board by phone at (512) 471-8871 or email at orsc@uts.cc.utexas.edu.

UT Austin IRB Approval Number: 2009-07-0046

A.1.3 Survey Text

With this survey we hope to find what people expect from a smart home (also known as an aware or intelligent home) and how they would like to "use" the home. A smart home is a residence that can monitor the environment and perform actions that would otherwise be performed by a human.

There are no right or wrong answers, and this is not a test of your abilities. Your answers will guide future research in aware homes.

Thank you very much for your participation!

1. How many years old are you?
2. What sex are you?
3. How would you rate your technology expertise?
4. Do you have children?

In this question we are interesting in learning how people would think about setting their thermostat.
Imagine you can monitor the conditions in your home. You can create policies that automatically perform actions based on the conditions.

There are devices in place to monitor

- the cost of energy
- date and time

In each room you can monitor the following information in your home:

- temperature (in degrees Fahrenheit)
- whether an inhabitant is present in the room
- a light level
- if there is motion
- the appliances that are on
- the current state of doors and windows

You can also monitor each inhabitant of the home, including pets. You will know their

- location
- activity (sleeping, standing, walking, etc)

1. Given the above list of sensors, what policy/policies would you use to control the thermostat? Please note that in this scenario, individual rooms or the entire home can be controlled. An example policy is to keep the temperature in the master bedroom at 72 degrees.

   There are no right or wrong answers and no length requirements; just write what you think.
In this question we will determine how people would detect and be alerted of a home invasion.

Now imagine you want to guard against burglaries. You have the same sensing capabilities as described above:

- the cost of energy
- date and time
- temperature (in degrees Fahrenheit)
- whether an inhabitant is present in the room
- a light level
- if there is motion
- the appliances that are on
- the current state of doors and windows
- location
- activity (sleeping, standing, walking, etc)

In this scenario you can also remotely control an array of items.

- You can control appliances in your home. For example, lights can be turned on or off, the washing machine can have it’s settings changed, or the thermostat can be set to a specific temperature.
- You can trigger audio devices such as buzzers, bells, and alarms
- You can also choose to send yourself email or text messages (SMS).
1. How would you choose to be informed of the suspicious activity?
   There are no right or wrong answers; write as little or as much as you like.

2. What policy would you create to detect that someone has broken into your home?

   In this question we hope to find what people want a smart home to do for them. Answers can be completely outlandish, practical, or anywhere in between. This is not specific to the previous thermostat and burglary examples. The home can monitor anything, including, but not limited to, the examples from before:
   
   • the cost of energy
   • date and time
   • temperature (in degrees Fahrenheit)
   • whether an inhabitant is present in the room
   • a light level
   • if there is motion
   • the appliances that are on
   • the current state of doors and windows
   • location
• activity (sleeping, standing, walking, etc)

• PLUS anything else you can think of

And the previously described controls:

• appliances in your home

• audio devices such as buzzers, bells, and alarms

• email or text messages (SMS)

• PLUS anything else you can think of

Here are two examples policies. 1) I would use an audio gauge, a camera, and a buzzer. The audio gauge and camera would be mounted over the backdoor. The buzzer goes off and the camera records the scene whenever the dogs make too much noise.

   2) Using a presence sensor on my cat and automated blinds, I would open the blinds whenever the cat is in the room.

   1. Given complete control of your home, what policies would you create?

   That completes our survey. Thank you very much for your participation!

   1. Do you have any feedback about the survey or smart homes?

   2. If you would be interested in further studies in this dissertation, please provide your email address in the space below. Note: Your email address will not be associated with your responses.
Bibliography


[70] D. Suther. X10 Protocol by Dan Suther.


