Designing Lightweight Software Architectures for Smart Environments

James Kusznir and Diane J. Cook
School of Electrical Engineering and Computer Science
Washington State University
Pullman, Washington, USA
(kusznir,cook)@eecs.wsu.edu

***Abstract*—Smart environment applications have gained a lot of attention and acceptance from the community. For this reason, many design and evaluation efforts target these applications. However, these applications rely on a software architecture that driven by a well-designed middleware. In this paper we propose design and evaluation requirements for smart environment software architectures and demonstrate how these requirements can be met with a simple, lightweight publish-subscribe design paradigm. We describe our CLM middleware that follows these requirements and illustrate its extensive use as part of the CASAS smart home system.**

***Keywords*-SmartHome Middleware, XMPP**

I. Introduction

Modern smart environments use a collection of sensors, processors, and control devices to allow a home or other space to interact with the resident in a productive manner. This has been a hot area of research for some time, yet designing well-integrated smart homes that are easy to use and deploy still eludes researchers. One of the most common problems with these smart environments is the integration of many varied components. Usually, a smart environment utilizes some off-the-shelf hardware systems from different vendors and sometimes some special-design hardware. These heterogeneous hardware components need to interoperate with one or more pieces of customized software to achieve the goals of the overall system. However, there is no ready-made common language, protocol, or other means of plugging them in and facilitating communication between these components.

The difficulty of creating lightweight, easy-to-use software architectures is one reason why few very fully-operational smart environment testbeds exist. When users read about or experience smart environment technology, their attention is drawn to the obvious benefits that such environments provide including automation, health monitoring, and context-aware assistance. For this reason, much of the design and evaluation effort is directed toward these applications. Behind these applications lies the smart environment software architecture, driven by middleware. In order for the smart environment applications to be robust, flexible, and efficient, the designer needs to impose such requirements on the middleware design.

Smart environment middleware connects the software and hardware components. As shown in Figure 1, middleware facilitates communication between the components and acts as the central "brain" of the system that collects data, messages, and information between the components. In this paper we provide a foundation for designing, implementing, and evaluating the middleware that drives smart environment architectures. Our goal is to design a middleware for smart environments that is lightweight, flexible, fast, and easily extensible. We hypothesize that this goal can be met by leveraging the power and simplicity of eXtensible Messaging and Presence Protocol (XMPP) using a publish-subscribe design paradigm. In this paper we propose design requirements for smart environments that provide a basis for middleware design and evaluation. We describe our middleware implementation, called "CASAS Lightweight Middleware (CLM)", which follows these requirements and has been successfully used to integrate multiple heterogeneous hardware and software components in CASAS.

II. Related Work

A number of researchers have proposed software architectures for smart environments but many of these proposals are not accompanied by implementation and evaluation. In contrast, however, very few of the fully-implemented smart environment projects publish details of their middleware or communication infrastructure. Hiding such details is consistent with the goal that middleware should simplify a project and enable the focus to be placed on the project's main goals and applications. There are a few existing middleware implementations that have been described in the literature. The University of Deusto SMARTLAB [1] utilizes OSGi [2] as a middleware framework. OSGi actually assists in creating middleware and authors need only to code the application logic. Since OSGi is designed for Java, the smart environment components need to reside on a single machine, be written in Java, and run under a single instance of JVM. Although OSGi facilitates gateways to other communication mechanisms such as HTTP, these extensions support a more limited set of functionality. UTA's MavHome project [3] designed middleware using the CORBA [4] middleware framework. In contrast with OSGi, CORBA supports communication across computers over a variety of networks and

- Simple – Easy to learn and use, not require specialized knowledge or training to grasp
- Light Weight – Minimize dependancy requirements and be capable of running on small, embedded hardware
- Capable of accommodating both known and unknown data
- Language and operating system independent
- Highly reliable – our smart homes are 24/7 production enviornments
- Maintainable over the long term – As the core of our infastructure, it will long outlast any group of students

Table I
SMART ENVIORNMENT MIDDLEWARE REQUIREMENTS.

has a rich feature set as well as open source implementations. Similar to CORBA, SOAP [5] provides a means of calling objects on remote servers and returning the results. SOAP's messages utilize XML and rely upon remote procedure calls or HTTP instead of a specific transport. Both of these frameworks are heavyweight, tightly-coupled systems which makes it undesirable for many smart environment systems. In an approach similar to the one described in this paper, Ristau [6] designs a middleware using a publish-subscribe framework. Because the architecture components are decoupled, the approach offers greater flexibility than earlier approaches, although it is still not as flexible or lightweight as the middleware design for CLM.

III. MIDDLEWARE DESIGN REQUIREMENTS

Occam's Razor [7] argues that the simplest theory is often the best. When we apply this basic principle to middleware design, it suggests to remove the many layers of indirection and protocol conversions, point to point connections, and other non-essential features that are found with other approaches and replace them with a simple, lightweight solution. Some of the naturally-evident features of a middleware that is designed with this approach are in line with features that are designed for smart environments. We propose the core requirements for smart environment middleware summarized in Table I.

Our design approach breaks a smart environment into modules we call "agents" plus a middleware "hub". The agents may be publishers, subscribers, or both (See figure 1. By making this division, all information specific to a task or sensor is contained in a single component, with communication between components happening in a standardized way. In software engineering, this is referred to as cohesion: All pieces of software associated to a responsibility are packaged together. Middleware should allow modules to be independent without requiring knowledge of other modules. This is referred to in software engineering as low coupling. Modern software engineering practices stress that good design approaches both of these principles and results in a far more manageable project [8]. By providing a lightweight middleware archetecture, these principals naturally flow in the remaining development.

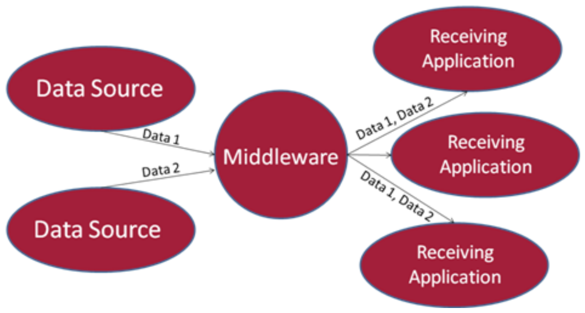

Figure 1. Where middleware fits.

IV. CLM

A smart environment system needs a means by which to send information between system components. CASAS enviornments contain several data sources (e.g., sensor hardware) and several applications that act as data receivers (e.g., AI algorithms, data architectures, graphical visualizers). As a result, CASAS enviornments are best served with an event-passing infrastructure. In their simplest form, these events consist of a data originator (such as a sensor), a timestamp, and a message. Using this format, virtually any sensor type can be accommodated and the middleware only needs to receive and re-transmit this message to all interested entities. We hypothesize that CLM can accomplish the middleware design goals by using a publish-subscribe mechanism for conveying data from sources to receivers. Here we provide details for the CLM middleware design and implementation.

A. Data format

Following the requirement of simplicity, we impose as few constraints on the spec as possible. To accomplish this, CLM uses XML for data formatting. XML's full text tags make it simple for humans or automated agents to extract the data. XML is open and flexible: any component can define XML schema tags and unrecognized tags are simply ignored. Table II shows CLM's XML messages. To ensure that all events are ordered by their timestamp, CLM uses a single clock to timestamp all messages. Next, every device, entity, or agent must have a globally-unique ID. The spec also includes a location for each entity which maps the physical location of the entity to a logical position. If no physical location is available then the agent can set the ID and location to be the same. Finally, the spec includes the message itself. This is a free-form text field with no formatting requirements. By not imposing such requirements CLM allows the addition of new sensors with previously-unanticipated data types.

B. Message transport

Exchanging messages between architecture components can be visualized as similar to humans sending an "instant message" on the Internet using a mechanism such as Google Talk. For this reason CLM employs instant message clients.

```
<publish>
  <channel>rawevents</channel>
  <data>
    <event>
     <serial>12C67946000000DB</serial>
      <location>M003</location>
      <message>OFF</message>
      <epoch>1271192051.34</epoch>
      <type>motion</type>
      <by>OneWireAgent</by>
      <category>entity</category>
      </event>
  </data>
</publish>
```

Table II
XML MESSAGE.

To meet with requirement of usability CLM uses an open source instant messaging platform that is friendly to XML, the XMPP (eXtensible Messaging and Presence Protocol, also known as "jabber"). XMPP has gained widespread popularity and thus also offers an advantage of a large collection of readily-available servers and client libraries.

1) Security: Maintaining the security of information gathered in a smart environment is essential for maintaining the privacy and security of its residents. By choosing XMPP is a transport, CLM can make use of open source XMPP servers which include security mechanisms as well as SSL and password protection. More importantly, the security mechanisms are designed and implemented by individuals with more expertise and focus in security than smart home middleware designers are likely to have. In addition, open source projects receive more open review of coding bugs that can impact security, and are more likely to have fewer bugs than in-house code. As all of our network traffic is handled by XMPP, this addresses our security needs.

C. Commercial implications

Smart home middleware design decisions also have interesting implications for future homeowners if CLM is used in commercial deployments. Because CLM uses all open standards it allows hobbyist smart home owners or commercial end users to easily integrate their own programming into their smart home and to monitor their smart environment. In addition, by using open source tools at the core, users and homeowners can inspect the code themselves for back doors or other vulnerabilities.

Our CLM implementation consists primarily of a manager daemon that listens on a known address for XML commands and data messages. The manager maintains a list of subscribers for different channels. When a publisher publishes information to a channel, it is simply sending the message to the manager. The manager then re-sends it out to all subscribers on that channel. All messages are passed as standard XMPP messages (the same as a human user sending a message to a friend via XMPP).

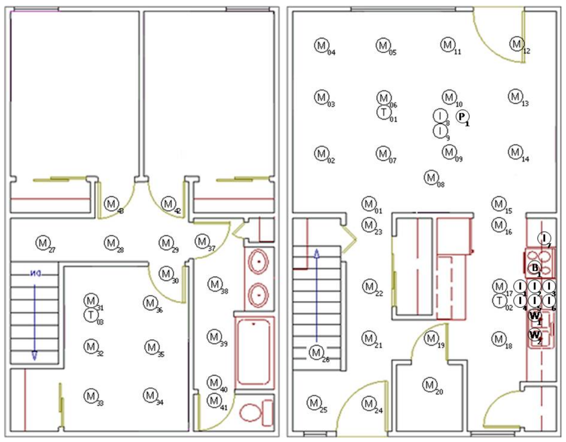

Figure 2. Diagram of our first living environment and sensor locations. Circles with M inside them represent a ceiling-mounted motion detector, T's are temperature sensors, and I are item sensors. The subscript refers to its local id number.

V. EVALUATION

In this paper we propose middleware requirements and describe the design and implementation of the CLM middleware that is consistent with these requirements. Here we also suggest that middleware can be evaluated in light of these same requirements and offer an evaluation of the CLM middleware along these performance dimensions. We evaluate our CLM middleware via our implementation that supports the CASAS smart home environment [9]. The goal of the CASAS project is to design smart environments that act as intelligent agents, perceiving the state of the environment using sensors and acting on the environment to achieve user or project goals [10]. The current CASAS environment consists of sensors connected via a Dallas OneWire bus. Additional sensors act in standalone mode or are connected through a customized interface (typically serial, USB, or network) or have their own bus. The CASAS system has been deployed at two long-term testbeds and in the homes of several volunteer study participants. A diagram of sensor locations in one of the CASAS testbeds is shown in Figure 2.

A. Simple

CLM has been simple to implement and maintain. The first version of CLM was written in a single weekend and immediately took over all middleware duties at the CASAS smart home testbed. This implementation is written completely in Perl. CLM has been operating continuously for over two years at the long-term testbeds as well as for shorter durations in participant homes.

B. Lightweight

The entire CLM middleware suite, including ejabberd, runs on a sheeva plug computing device [11], which is an embedded computer the size of a "wall-wart" style power converter that consumes a miserly 4W of electricity (meeting the "small" middleware design requirement). The speed requirements for CASAS, are to process a minimum of 10 messages per second, with 20-30 begin preferred. CLM easily met this requirement on the sheeva plug hardware.

C. Independent

CLM is universally compatible and has supported components on multiple operating systems and software agents written in multiple programming languages. To facilitate the distribution of agents across multiple computers we need to ensure that bandwidth use is minimized. XMPP-based middleware will in theory require more bandwidth due to its use of XML over a dense binary format. To test XMPP for this requirement and compare it to CORBA, we ran the publisher and middleware core on one computer with a subscriber on a second computer. We found that CORBA is smaller due to its binary protocol (although a number of debugging issue arose which were much more difficult to address with CORBA). However, as the number of subscribers increases a difference in performance is apparent and is graphed in Figure . While XMPP provided near-constant delivery times, CORBA grows linearly with the number of subscribers.

D. Usable

CLM has been running on a variety of lightweight and embedded computers. A half dozen students and researchers have used and contributed to the middleware without a significant learning curve. For a sampling of the agents created using CLM, see below.

E. Expandability

Perhaps the greatest testament to the usability and expandability of CLM is the number of components that have been successfully used, integrated, tested and deployed as part of CASAS that rely on this middleware. Some example components are two different power meter interfaces (OneMeter and TED), two different graphical event visualizers, and an online AI agent that not only subscribes to events but also performs computations on them and publishes conclusions on a separate channel. We have also leveraged the flexibility to replace core agents such as our database logger. This transition was clean and seamless, requiring no downtime or modifications to other agents. A list of agents that have been written using CLM is provided below.

1) OneWire. This is the first of the CLM agents. It is a publish-only agent that manages the OneWire bus, our primary sensor platform. This is where are motion detectors, door/cabinate detectors, temperature sensors, and basic analog value reading sensors live. All of our deployments to date consist of a OneWire bus and this agent.
2) DBLoader. This subscriber-only agent logs collected data to a local PostgreSQL database. As the project requirements evolved, it was deturmined that the local SQL database was not required, and it was rewritten to log into a flat text file and upload them to our central server on a regular interval. This version of this agent is called Scribe, and is currently in use by CASAS.
3) Insteon. The CASAS testbeds employ Insteon light controllers which represent our primary control system. This agent is both a publisher and a subscriber and is written in Perl.
4) OneMeter. The CASAS project collects electricity usage using either a OneMeter or TED meter. The OneMeter and TED meters are publish-only agents that report power consumption in watts for an entire smart environment testbed.
5) Chumby. The CASAS project uses a small touchscreen computer known as a "Chumby" to provide interaction with environment residents. The CASAS Chumby manager agent sends messages via the Chumby channel to instruct the Chumbys what content to present to the user and collect user input.
6) Sensor Event Visualizer. To better understand events in the CASAS testbeds and patterns of resident behavior, CASAS researchers created visualizers to graphically display sensor events. These visualizers included a raw ASCII visualize, a 3D model built on SecondLife, and the PyViz 2D visualizer, shown in Figure 3. These are subscriber agents that are heavily utilized by environment residents, by personnel who annotate collected data with activity labels [12] [13], and by researchers.
7) Prompter. In one of the current CASAS studies, older adults with dementia are provided context-aware prompts to reminder them to initialize important daily activities. In this study a Asus EEE-Top touchscreen system issues audio and pictorial prompts and accepts input from the participate to indicate whether or not the activity was completed. This agent is thus both a publisher and a subscriber that relies on CLM middleware

VI. Conclusions

In this paper we propose requirements for smart environment middleware. We also described the CLM middleware that is designed to meet these requirements and evaluated CLM along these performance dimensions. We demonstrate that using XMPP as the foundation of the CLM allows the performance requirements to be met. Not only is CLM theoretically useful, but it is also in active use in the CASAS project, where it has virtually limitless uptime (the current uptime record is over 200 days with 24/7 data collection in an inhabited space). We will continue to add enhancements for security. We are also interested in "linking" smart spaces rather than treating them as independent entities. Future versions of the middleware will also consider ways of further

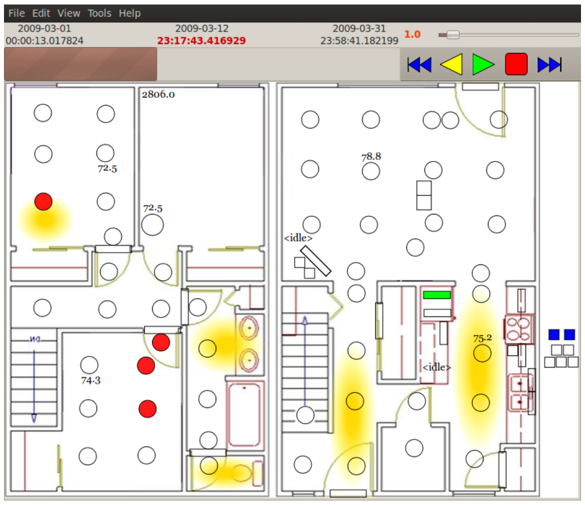

Figure 3. PyViz screenshot.

improving middleware performance, knowing that improves at the middleware level result in significant benefits to the entire smart environment system, its applications, and its users.

References

[1] J. I. V. D Lopez-de Ipina, A Barbier, “Dynamic discovery and semantic reasoning for next generation intelligent enviornments,” in *Proc. IE’08*, 2008.

[2] (2010) Osgi alliance. [Online]. Available: http://www.osgi.org/Main/HomePage

[3] G. M. Youngblood and D. Cook, “Data mining for hierarchical model creation,” *IEEE Trans. Syst., Man, Cybern. C*, vol. 37(4), pp. 571–572, 2007.

[4] (2010) Object management group, coordinating body of corba homepage. [Online]. Available: http://www.omg.org/

[5] (2010) Soap specification. [Online]. Available: http://www.w3.org/TR/soap/

[6] H. Ristau, “Publish/process/subscribe: Message based communication for smart environments,” in *Proc. IE’08*, 2008.

[7] R. Epstein, “The principle of parsimony and some applications in psychology,” *Journal of Mind Behavior*, vol. 5:199-130, 1984.

[8] S. L. Pfleeger and J. M. Atlee, *Software Engineering: Theory and Practice 4th Ed.* Prentice Hall, 2010.

[9] P. Rashidi and D. Cook, “An adaptive sensor mining model for pervasive computing applications,” in *Proc. of the KDD Workshop on Knowledge Discovery from Sensor Data*, 2008.

[10] D. Cook and e. S. Das, *Smart Environments: Technologies, Protocols and Applications*. Wiley, 2004.

[11] M. Michlmayr. (2009) Sheevaplug: the nslu2 killer. [Online]. Available: http://www.linuxtoday.com/news_story.php3?ltsn=2009-03-19-008-35-OS-HW-DV

[12] B. M. S. Szewcyzk, K. Dwan, “Annotating smart environment sensor data for activity learning,” *Technology and Health Care, special issue on Smart Environments: Technology to support health care*, vol. 17, pp. 161–169, 2009.

[13] G. S. D. Cook, A. Crandal and B. Thomas, “Detection of social interaction in smart spaces,” *Journal of Cybernetics and System*, vol. To appear.