SenseWrap: A Service Oriented Middleware with Sensor Virtualization and Self-Configuration

Pål Evensen, Hein Meling

Department of Electrical Engineering and Computer Science, University of Stavanger
4036 Stavanger, Norway
[1] paal.evensen@gmail.com
[2] hein.meling@uis.no

***Abstract*—This paper presents the design and implementation of a simple and elegant middleware architecture providing *virtual sensors* as representatives for any type of physical sensors. With our middleware, clients can seamlessly discover sensor-hosted services through Zeroconf and it provides a standardized communication interface that applications can use without having to deal with sensor-specific details. The limited capabilities of most types of sensors prevent the inclusion of a full communication stack with IP addressing. Yet, through the use of virtual sensors, a uniform communication interface based on UDP/TCP sockets can be exposed to clients. This will significantly simplify application development for integrated services involving multiple types of sensors. Our benchmarks shows that our middleware scales well beyond the requirements of a private smart home.**

I. Introduction

Wireless communication technologies enables seamless communication between residential network entities such as set-top-boxes, sensors, control units and other devices, and are typically far less costly to install than their wired counterparts due to cabling. These technologies have opened up a whole range of new applications in the utility segment, like remote control of heating, security and safety systems and health monitoring.

However, the heterogeneity of communication protocols and the mixture of addressing schemes used by networked devices of different make and model is one of the biggest challenges when developing integrated smart home services. Most smart home systems offered today are based on proprietary all-in-one solutions, where the sensors and actuators might use a proprietary RF protocol over the 868MHz band, while others might use ZigBee, Bluetooth or WiFi. Furthermore, most devices have their own application-level protocol for communicating control commands and retrieving data. Moreover, due to the limited capabilities of many types of sensors, a full communication stack with IP addressing is simply unfeasible. Yet, it would significantly simplify application development if interaction with the sensors were based on UDP or TCP sockets and IP addressing schemes. Currently, these issues hampers innovation and development of new (possibly third-party) smart home services. Another obstacle to the adoption of smart home technology is the complexity of setting up and managing the networking between devices, deterring most home owners from acquiring such solutions. Hence, it is paramount to the success of networked homes that device configuration is performed automatically.

This paper builds on previous work [6] and presents the design and implementation of a simple, yet elegant middleware architecture providing *virtual sensors* as representatives for any type of physical sensors. Improvements over the previous version includes a cleaner, more generic system architecture and added support for publish/subscribe style of interaction. Additionally, we have included a performance evaluation, which was missing in the previous paper. Our middleware, which we have named *SenseWrap*, combines the Zeroconf protocols with hardware abstraction, giving a service-oriented and lightweight middleware for application programmers to interact with. A virtual sensor provides transparent discovery of arbitrary sensor devices through the use of Zeroconf protocols [3]. This enables applications to discover sensor-hosted services through Zeroconf and it provides a standardized communication interface that applications can use without having to deal with sensor-specific details. That is, virtual sensors also provides a uniform communication interface to clients, based on UDP/TCP sockets or even HTTP. This is accomplished by abstracting functionalities common to most sensor models, and writing custom wrappers (drivers) for the specifics of each sensor model. This way, applications need not know anything about the physical or logical communication protocols used by the sensors, making the same network services usable with any sensor model sharing the same basic functionality. For instance, a light-controlling application should be able to operate independently of the actual luminosity sensors used. Note that the architecture is generic and can be used in a wide range of application areas where sensors needs to be connected; however, for the sake of illustration, the examples presented here are framed in a smart home setting.

By using virtualized sensors, third-party developers do not need to learn any custom sensor APIs to interact with the sensors, even though the capabilities of the sensors are limited to low-level RF communication. Assuming sensor vendors provide the sensor communication API, third-party developers can supply the necessary custom wrappers for the middleware to use, or vendors can provide such wrappers. Virtual sensors

gives flexibility to applications, since replacing sensor devices does not require modifying the implementation of applications using those sensors. This is assuming the basic interaction is the same or similar. Furthermore, with technology innovation, new sensor models may natively support Zeroconf and link-local IP addressing. Applications can then use these with minimal changes, bypassing the virtual sensors.

The rest of this paper is organized as follows: Section II presents the background for the paper, gives an overview of related work, and state our assumptions. Section III presents the architecture of our middleware for virtualized sensors, and Section IV provide some relevant implementation details. Section V presents and evaluates test results and Section VI concludes the paper.

II. Background and Assumptions

The middleware focuses on self-configuration and will offer support for developing integrated services, where multiple services can interact to offer synergies across different technologies: For instance, a light-control service could interact with the movement sensors associated with the alarm service, in addition to luminosity sensors, to decide whether the light should be switched on.

In the context of the IS-home project, we assume a networked device capable of running our middleware; this could be a simple *embedded computer* running the Linux operating system, like a base station, router etc. Further, we assume the computer has multiple interconnection interfaces, e.g. ZigBee, Bluetooth, WiFi, GPRS, Ethernet and USB ports for connecting alternative network devices. This computer may run one or more network services, and may act as a gateway between different network applications and devices.

Zeroconf is an essential component in our middleware architecture; sensors and other network devices will be registered with and discoverable through Zeroconf. Hence, a brief overview of Zeroconf is given below.

A. Zero Configuration Networking

Zero configuration networking is endorsed by the Internet Engineering Task Force (IETF) [10], through various RFCs. Zeroconf automates three core services: IP addressing, name resolution and service discovery [3]. In other words, IP addresses will need to be assigned automatically to each device and coupled with a meaningful name, and services have to be discovered automatically as they enter the network. This is achieved with the following combination of techniques [3]:

- *Link-local addressing* is used to assign IPv4 addresses without relying on a DHCP server present on the network.
- *Multicast DNS (mDNS)* is used to provide name binding without a DNS server present.
- *DNS Service Discovery (DNSSD)* enable users to browse for services without having to know anything about the hosts providing them.

The philosophy behind the Zeroconf platform is rooted in the assumption that end users are interested in services, not devices. The goal is that users should be able to select services from a list through a graphical user interface.

Each Zeroconf-enabled device keeps its own list of services that is kept up to date in a distributed and thought-out manner, using a combination of techniques such as multicasting and polling to keep track of available services present on the network. Combined, these methods prevents the network from being flooded with control traffic.

Universal Plug and Play (UPnP) [7] is an open ended collection of protocols that offers some of the same functionality as Zeroconf. Our reason for going with Zeroconf is that UPnP is rather heavyweight, communicating with SOAP XML objects over HTTP and has a flawed service discovery protocol, built on an abandoned IETF draft [2]. Please see work [6] for a more detailed comparison between UPnP and Zeroconf.

B. Related work

In previous work, Construct [5] offers a distributed middleware for pervasive systems and provides mechanisms for capturing sensor data and converting them into RDF formatted data for storage. Like SenseWrap, Construct employs Zeroconf to locate services, but does not allow discovery of sensor devices as in our middleware. While our focus is on finding a standardized way for applications to communicate with sensors, the main focus of Construct appears to be on data capture and the processing of information. SStreaMware [9] is another middleware that shares some features with SenseWrap, but is an all-encompassing solution where sensor interaction is performed via a provided graphical user interface and not at application level. Our approach to virtualizing sensors based on Zeroconf, using *protocol adapters* to interface with applications is more lightweight and allows better application level adaptation.

Hourglass [12] is an infrastructure for connecting sensor networks to applications. It provides a *data collection network*, that aggregate functionality from several disparate sensor networks, and offer this to Internet-based applications. Compared to our architecture, Hourglass focuses on the underlying network links and data streams more than the service aspect. The main effort is on handling unreliable connectivity by providing links between networks and applications that buffers data and retransmits these at a later point in cases of link loss. Neither Hourglass or Global Sensor Network (GSN) [1] focus on service discovery. Like our own middleware, GSN aims to solve the problem of hardware heterogeneity in sensor networks. GSN also use *adapters* to abstract physical devices into virtual sensors. With SenseWrap, we take the abstraction one step further by virtualizing the services as well. With GSN the emphasis is to provide the ability to query all supported sensors using SQL, and to provide a homogeneous view of sensor data.

Open Services Gateway initiative (OSGi) provides a gateway for connecting different devices and services together through a central point, allowing applications to be composed from different, reusable service modules [4]. The framework is module based and only specifies the application programming

interface, not the underlying implementation, leaving it up to the developers to handle the actual communication with the sensors or actuators.

Using OSGi as a foundation, Gürgen et al take a "database approach" in their SStreaMWare middleware [9], offering a schema to represent sensor data in a generic manner. Interaction with the sensors is performed with declarative queries in a SQL-like relational language. Like SenseWrap, SStreaMWare uses *adapters* to transform generic commands into the necessary device-specific format, and it also provide both publish/subscribe and request/reply communication models. However, the scope of SStreaMware is quite different from SenseWrap, as SStreaMWare comes as a complete package, where sensor interaction is performed via a provided graphical user interface and not at application level. This makes the system difficult to adapt to third party applications, which it is clearly not intended for. The scope of our middleware is to facilitate integration between sensors and applications with minimal effort. Our approach to virtualizing sensors based on Zeroconf, using *protocol adapters* to interface with applications is more lightweight and allows better application level adaptation.

Tenet [8] is more of a network architecture than middleware, dividing sensor networks into tiers, consisting of masters and motes. The argument for this architecture is that sensor motes are unreliable and underpowered, hence all but the simplest computing tasks are better left to more powerful master nodes. Furthermore, the authors claims that software re-usability is enhanced by having most of the application logic on master nodes, as device specific customization of the code is less likely to be needed. This is not unlike our approach, but instead of several masters, we use a single gateway to perform the heavy lifting in terms of computational tasks. The reason for not using several masters is simply that we don't see the need for more in a private smart home, although it would be relatively easy to include additional gateways if required (one way of achieving that would be to set up an additional gateways to listen for different types of services).

III. Architecture overview

The middleware architecture is organized into multiple layers of abstraction to provide sensor-based services to clients. That is, physical sensors appears to behave as if they provide Zeroconf-like services. Hence, the services provided to applications become independent of the sensor hardware. The middleware takes advantage of standardized Zeroconf protocols to provide automatic network configuration of sensors and service discovery to clients. This makes the sensor services available to any Zeroconf-enabled application on the same network.

Our middleware define two core entities: The **Sensor Unit** is a virtual representation of the physical device hosting the actual sensors and actuators. Attributes include identity (typically a MAC address) and location. Sensor units are subclassed into sensor types such as Sun Spots, SquidBee, etc. It is the implementation of a Sensor unit that handles the communication between the middleware and the actual sensor.

A **Service** is hosted on the sensor unit, and can either be a detector or an actuator. Examples of detectors include sensors for temperature, humidity and luminosity. Examples of actuators are power and light switches, thermostats and locking mechanisms.

A. Application Protocol

SenseWrap supports both the request/reply and publish/subscribe communication model. The default is request/reply with the subscribe model available through additional parameters.

After a service has been looked up through Zeroconf, and connection has been established, the client applications use generic commands to communicate with the services.

For instance, the way to do a simple temperature reading would be issuing the command GET to the service. This would return a single reading. If the client wants to subscribe to the temperature service, it can ask the middleware to feed it with periodic readings by appending the keyword SUB followed by the desired interval in milliseconds. The middleware will keep sending readings at the specified interval until it receives a STOP message, or until the connection is closed.

The main components of the middleware are:

- **DiscoverSensors** listens on the network for new sensor devices, and creates virtual representations of these.
- The **Sensor** class communicates directly with the sensor nodes, and keeps track of connectivity. It translates application commands received through the protocol adapter and forwards these to the physical sensor, using the native communication protocol of the sensor.
- A virtual **Service** represent a service provided by a sensor. It registers the communication endpoint (host name and port number) of the service with Zeroconf and listens for connection requests from clients. Upon receiving a connection request, the service creates a protocol adapter to handle the communication with the client.
- Clients communicate with sensors through **Protocol Adapters**. They provide a standardized communications interface independent of the kind of sensor involved in the communication, and are generic for all services. Once the application has established a connection with the protocol adapter, the adapter communicates directly with the virtual sensor.

The *DiscoverSensors* and *Sensor* implementation are the only components in our architecture that needs customization. That is, they are both comprised of a generic part, and a custom part that needs to be tailored specifically for each supported sensor type. Keeping in line with the service-oriented philosophy of Zeroconf, our middleware separates the services from the sensors. This is the most flexible solution as it allows the system to support more than one service per sensor, e.g. a single sensor unit may contain both temperature and humidity sensors. The separation of services from sensors adheres to established

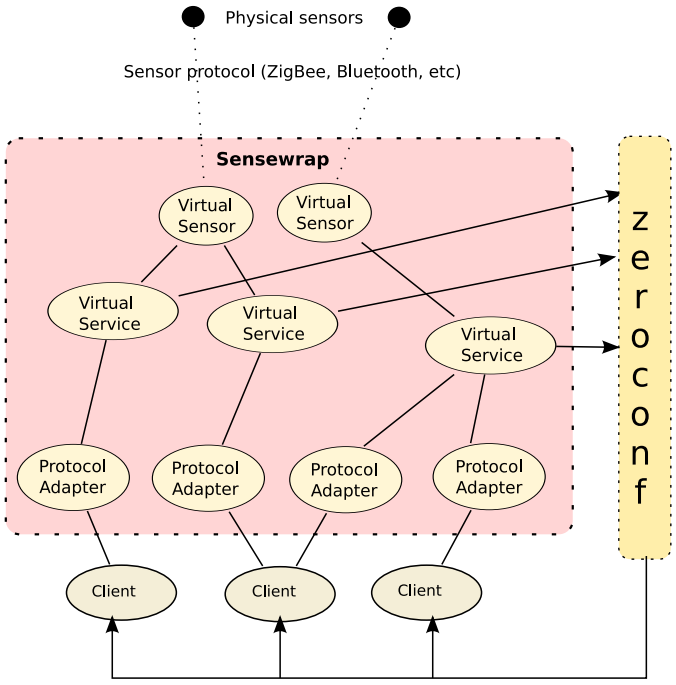

Fig. 1. Detailed middleware architecture

object-oriented principles, as it promotes high cohesion and low coupling between components. The details of a sensor's physical connection and battery status does not logically relate to the attributes of, for instance, a temperature service. For the same reasons, the protocol adapters are separated from the virtual sensors and virtual services, as the connection details between applications and virtual services are neither related to the logic of the sensor nor the service.

Having the services separated from the sensors allows the service component to be generic for all supported sensor types. In addition, this approach is a good match with the Zeroconf APIs, as the methods provided by these are geared towards services instead of devices.

Figure 1 show a conceptual view of the system. Each physical sensor is represented by a corresponding virtual sensor. Furthermore, each service offered by the sensor is represented with a virtual service. A virtual sensor can have many services, e.g. if the same physical sensor device is a multi-sensor device, the different sensor readings can be offered to applications through distinct virtual services. A virtual service can also have many connections through different protocol adapters. For example, multiple services for the same sensor can be registered with Zeroconf at the same time, one accessible over TCP and another over SOAP.

Client applications use Zeroconf to identify and locate services provided by sensors, and communicates with them through the protocol adapter. An application can be composed of one or more services, but only needs one socket per service.

IV. Middleware Implementation

SenseWrap is written in Java, with the core components represented in the classes *DiscoverSensors*, *Sensor*, *Service*, *ClientHandler* and *BonjourRegistration*.

DiscoverSensors maintains a list of sensors that the middleware is capable of communicating with. As the sequence diagram in Figure 2 illustrates, the *DiscoverSensors* listens for service advertisements broadcast by sensors in the network.

After the service has been registered with Zeroconf, it listens for socket requests on the corresponding TCP port, and spawns a *ClientHandler* thread for each connection request.

Clients can multicast a DNSSD request for available services that resides on the same network and the Zeroconf framework will reply with the name of the host on which the virtual service is running, and the port number which to connect to. An application can then send a connection request and get a TCP socket in return. Commands received by the client handler is forwarded to the virtual sensor, which translates these into the appropriate sensor-specific command, which, in turn, is transmitted to the physical sensor, using the device's native communication protocol.

Each virtual sensor keeps track of the state of its associated physical sensor. A sensor is considered to have failed if an *IOException* is caught, e.g. due to a communications failure. If a sensor fails, the virtual sensor is responsible for *unregistering* the service from Zeroconf, removing itself from the list of sensors maintained by the *DiscoverSensors*, and terminate. Similarly, if an *IOException* is caught when clients are trying to access the service, the virtual service will be unregistered from Zeroconf itself.

A. Adding New Sensor Types

Adding support for new types of sensors involves developing device-specific versions of the *DiscoverSensors* and *Sensor*. In order to simplify development, the middleware comes with abstract versions as well as interfaces for these components, allowing implementations to reuse common functionality, effectively giving developers a blueprint of the required classes.

Essentially, the custom part of the service factory needs code for detecting connection requests from the physical sensors and for creating the appropriate virtual sensor. Obviously, the virtual sensor must also be able to communicate natively with the physical sensors.

B. Adding New Communication Protocols

The protocol adapter is a generic communication interface through which clients connect. Different applications might require different communication protocols, and the middleware supports adding new protocol adapters. Currently, a TCP-based protocol adapter is supported, while support for UDP, HTTP, SOAP and RMI can easily be added, as shown in Figure 3. Once an adapter has been developed, it can be reused without modification for all sensor types supported by the middleware. In addition to making the middleware flexible, this ensures future compatibility with new protocols as they emerge.

V. Performance

Because the middleware is intended to run on a dedicated machine within the home, we do not see scalability as a big

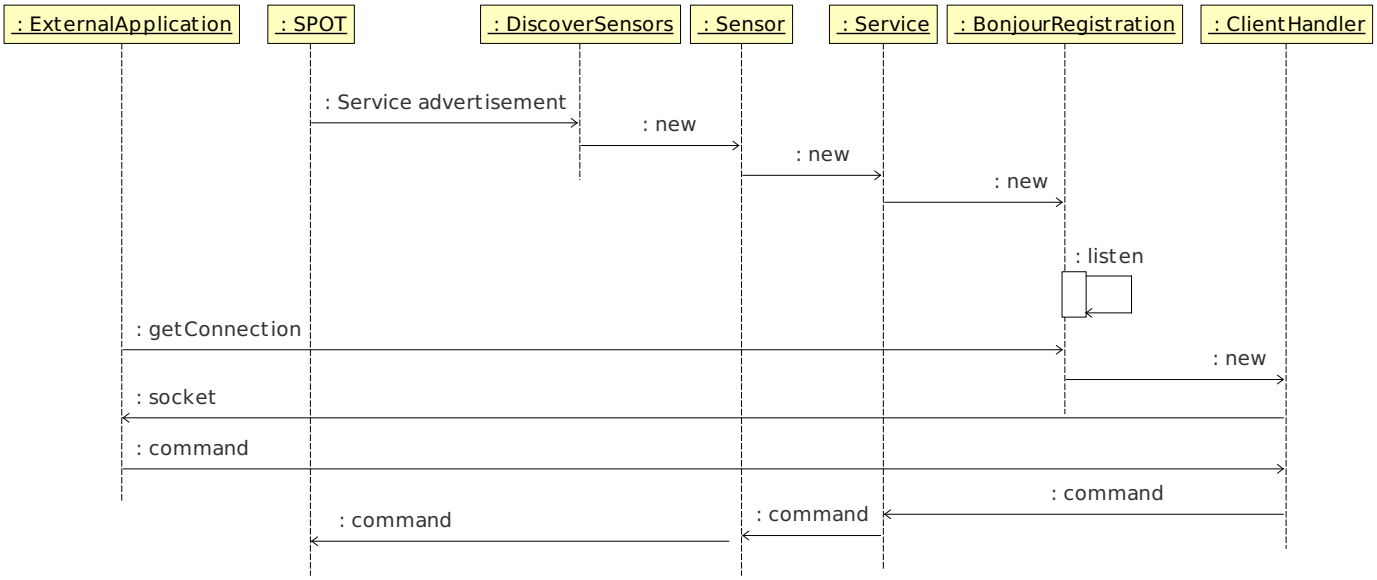

Fig. 2. Service discovery and connection establishment

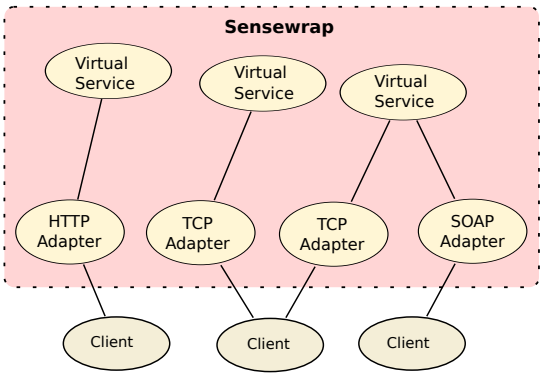

Fig. 3. Protocol adapter

concern. Typically, the number of sensors to be handled in such an environment are limited to less than 100, and as such, the demands for scalability is not critical. However, we have performed tests on this matter to reveal potential flaws of the architecture. Response time (the time it takes for clients to receive an answer to a request) under realistic client load was measured with clients

Regardless of the scalability of the middleware itself, the number of services running on the middleware is limited by the underlying Zeroconf framework, which becomes ineffective when the number of nodes approaches 1000 [3].

Tests were performed by running the middleware on a dedicated machine, while polling it for sensor readouts from other machines on the same local network. At the most, 19 computers, hosting eight clients each was continuously polling the middleware. The "server" had 2GB of RAM, an Intel Core Duo 2 E8300 processor and was running Fedora Core 11 with Sun's Java version 1.6_14. Measurements was made for the request/reply model.

Execution time elapsed between query and response from a temperature sensor on a Sun SPOT through the SenseWrap middleware was measured at the client. A caching mechanism implemented in the middleware was set to reread values from the sensor only if the existing value was older than four seconds.

A. Results

Performance were measured to an average of 6.8 milliseconds with a test run of ten simultaneous clients, each issuing 1000 requests (figure 4), immediately sending a new query as soon as a reply is received. This amounts to an average capacity of handling about 147 queries per second under load. Predictably, the average response times rise as more clients are jamming the middleware with queries, and drops to a capacity of around 5.5 queries per second with 152 simultaneous clients.

The scatter plot (figure 5) shows an excerpt of 14000 operations from a run of 95 clients simultaneously querying the middleware a total number of 190000 times while caching of sensor readings is set to four seconds. The plot starts ten seconds into the experiment, to be sure that all clients has started. The y-axis shows the round-trip time for each query, measured in seconds. The x-axis shows time elapsed, also measured in seconds.

An observation that can be made from figure 5 is that it takes only 6.38 seconds to finish 14000 operations, giving an average of 4.6 milliseconds per operation as opposed to the lowest average of 6.8 milliseconds measured for each single operation. This indicates that having the clients waiting for a response before issuing a new command does not load the middleware sufficiently to make it the performance bottleneck.

B. Evaluation

In a smart home scenario, the middleware is likely to run on less powerful hardware than what was used in our tests, but

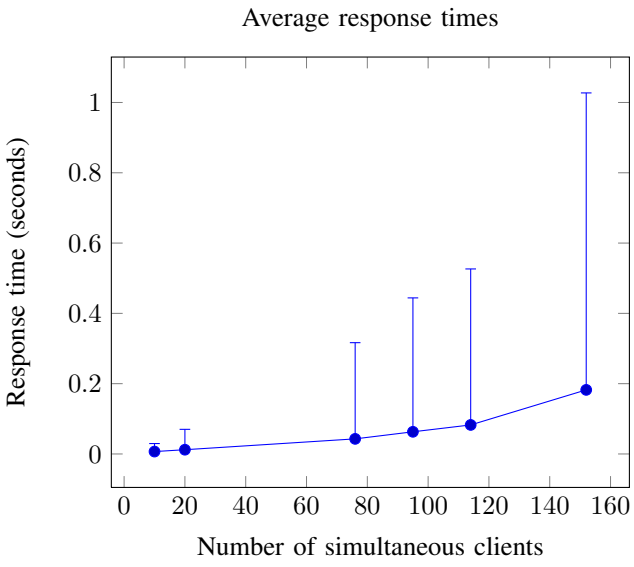

Fig. 4. Times measured at the clients

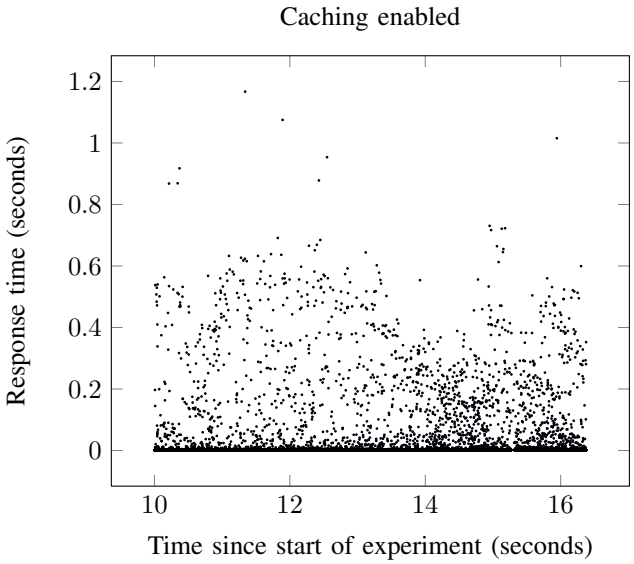

Fig. 5. Response times (95 simultaneous clients, 14000 requests)

our rationale is that even if one divide the performance by ten, it is still more than sufficient to handle the requirements of a typical smart home. We also measured the execution time for each renewal of the cache at the server. A total of 56 sensor readings had an average value of 951.57 ms, which would give the middleware a capacity of just over one reading per second per sensor, thus illustrates the performance gain of caching.

VI. Conclusions and Future Work

By virtualizing the physical sensors in smart homes, we can provide client applications with a uniform communication interface. We have demonstrated how the important task of automating the discovery of services and devices as well as the networking between applications can be solved using Zeroconf.

The need to implement routing in the middleware became apparent under testing, as values was not always returned to the correct client. This could be solved by tagging incoming requests with thread ID and looking up the correct thread before returning a value.

While the middleware presented here makes the communication protocol between sensors and application generic, the application protocol is not. By implementing an ontology built with OWL, the application protocol could be made generic and platform independent as well. We also intend to expand our application to include support for other types of sensors beyond the Sun SPOTs supported in the current implementation.

Enabling remote access to the services in the home over wide area networks such as the Internet or GPRS can be useful for tasks like adjusting the heat before coming home, or turning off the alarm to let someone in. Remote accessibility brings up some security and privacy concerns that needs to be addressed at some point.

Having multiple higher-level applications competing for resources (actuators) introduces the issue of resource ownership and dependency management. For instance, two applications accessing the same actuators could potentially result in conflicts where one of them is constantly turning a switch off, while the other turns it back on. A priority concept, like the one outlined by Retkowitz and Kulle [11] could be worth looking into in future versions.

References

[1] K. Aberer, M. Hauswirth, and A. Saheli. The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical report, School of Computer and Communication Sciences, Ecole Polytechnique Federale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland, 2006.

[2] S. Chesire. How does zeroconf compare with viiv/dlna/dhwg/upnp?

[3] S. Chesire and D.H. Steinberg. *Zero Configuration Networking - The Definitive Guide*. O'Reilly, 2006.

[4] P. Dobrev, D. Famolari, C. Kurzke, and B.A. Miller. Device and service discovery in home networks with osgi. In *IEEE Communications Magazine • August 2002*, pages 86–92. IEEE, 2002.

[5] S. Dobson, P. Nixon, L. Coyle, S. Neely, G. Stevenson, and G. Williamson. Construct: An open source pervasive systems platform. In *Consumer Communications and Networking Conference, 2007. CCNC 2007. 2007 4th IEEE*, pages 1203–1204. IEEE, 2007.

[6] P. Evensen and H. Meling. Sensor virtualization with self-configuration and flexible interactions. In *Casemans '09: Proceedings of the 3rd ACM International Workshop on Context-Awareness for Self-Managing Systems*, pages 31–38, New York, NY, USA, 2009. ACM.

[7] UPnP Forum. Upnp device architecture 1.0, 2008.

[8] O. Gnawali, B. Greenstein, K. Jang, et al. The tenet architecture for tiered sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems(SenSys'06)*. ACM, 2006.

[9] Gürgen, Roncancio, Labbé, Bottaro, and Olive. Sstreamware: a service oriented middleware for heterogeneous sensor data management. In *5th int'l conf. on Pervasive services*, Sorrento, Italy, 2008.

[10] IETF. Internet engineering taskforce. http://www.ietf.org/.

[11] D. Retkowitz and S. Kulle. Dependency management in smart homes. In Twittie Senivongse and Rui Oliveira, editors, *Distributed Applications and Interoperable Systems, 9th IFIP WG 6.1 International Conference (DAIS 2009)*, volume 5523 of *LNCS*, pages 143–156. Springer Verlag, 2009.

[12] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. Technical report, Harvard University, 2004.